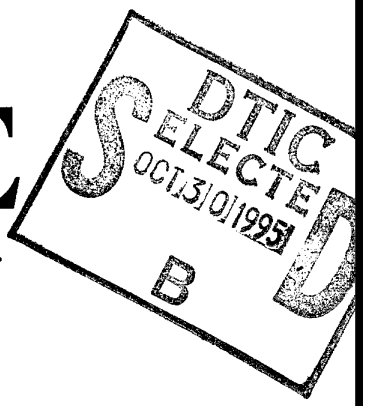


Proceedings of the

IFIP WG 11.3



Eighth Annual WORKING CONFERENCE on DATABASE SECURITY



Bad Salzdetfurth, Germany 23 - 26 August 1994

sponsored by

IFIP Working Group 11.3 **University of Hildesheim**

Editors

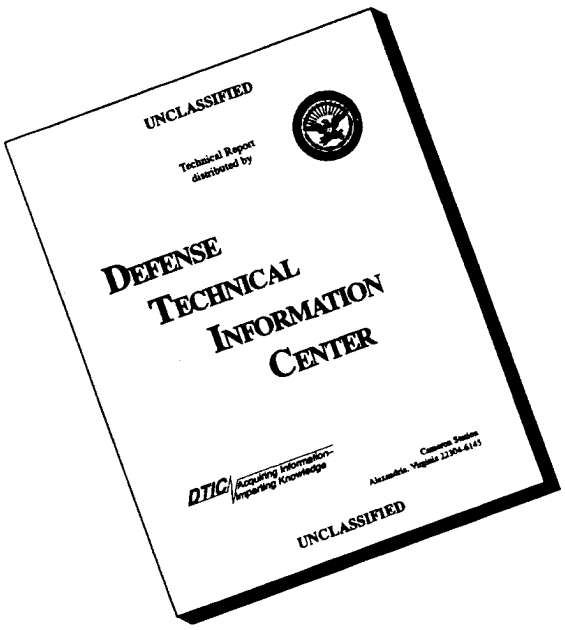
J. Biskup, M. Morgenstern, C. Landwehr

ISSN 0941-3014 Hildesheimer Informatik-Berichte 20/94, August 1994,
Universität Hildesheim, Institut für Informatik, 31141 Hildesheim, Germany

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

19951027 015

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Preface

These proceedings contain the papers presented at the Eighth Annual IFIP WG 11.3 Working Conference on Database Security held in Bad Salzdetfurth near Hildesheim, Germany, August 23-26, 1994. The conference has been sponsored jointly by the International Federation for Information Processsing (IFIP) Working Group 11.3 on Database Security and the University of Hildesheim. Furthermore the U.S. Office of Naval Research supported travel of many U.S. participants. As with its predecessors the purpose of the conference was to present original work in database security research and practice, to enable participants to benefit from personal scientific discussions and to expand their knowledge, to support the activities of the Working Group, and to disseminate its results.

All submitted papers were reviewed by working group members and a small number of external reviewers. Based on these evaluations 18 papers were finally accepted for presentation. In addition the program included a session on status reports on current projects and a panel discussion on perspectives on database security. These sessions are also documented in these proceedings by short abstracts.

We highly appreciated two invited lectures that are also included in these proceedings. Ab Bakker from BAZIS, Leiden, Netherlands, presents his deep insight into security in health care systems, and thereby we were able to continue the discussion of security in this important field of application. Klaus Dittrich from University of Zürich, Switzerland, challenges the security community with current trends in database technology to which he himself has substantially contributed.

A Working Conference is based on the joint efforts of many people. We thank all of them sincerely: the authors of submitted papers, the reviewers, the participants, in particular the invited speakers. We are also particularly grateful to Jimmy Brüggemann and Christian Eckert for local organization.

Program Co-Chair

Joachim Biskup
Institut für Informatik
Universität Hildesheim
Samelsonplatz 1
Postfach 101363
D-31113 Hildesheim
+49(5121)883-731 (voice)
+49(5121)883-732 (fax)
biskup@informatik.uni-hildesheim.de

Program Co-Chair

Matthew Morgenstern
Design Research Institute
Cornell University
5144 Upson Hall
Ithaca, NY 14853
U.S.A.
+1(607)255-9899 (voice)
+1(607)254-4742 (fax)
morgenstern@cs.cornell.edu

IFIP WG11.3 Chair

Carl E. Landwehr
Naval Research Laboratory
Code 5542
4555 Overlook Ave., SW
Washington, DC 20375-5000
U.S.A.
+1(202)767-3381 (voice)
+1(202)404-7942 (fax)
landwehr@itd.nrl.navy.mil

OFFICE OF THE UNDER SECRETARY OF DEFENSE (ACQUISITION)
DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22304-6145

IN REPLY REFER TO DTIC-OCC

SUBJECT: Distribution Statements on Technical Documents

TO: OFFICE OF NAVAL RESEARCH
CORPORATE PROGRAMS DIVISION
ONR 353
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

1995 1037 015

1. Reference: DoD Directive 5230.24, Distribution Statements on Technical Documents, 18 Mar 87.

2. The Defense Technical Information Center received the enclosed report (referenced below) which is not marked in accordance with the above reference.

CONFERENCE PROCEEDINGS
N00014-94-1-0830
TITLE: IFIG WG 11.3 WORKING
CONFERENCE ON DATABASE SECURITY

3. We request the appropriate distribution statement be assigned and the report returned to DTIC within 5 working days.

4. Approved distribution statements are listed on the reverse of this letter. If you have any questions regarding these statements, call DTIC's Cataloging Branch, (703) 274-6837.

FOR THE ADMINISTRATOR:

1 Encl

GOPALAKRISHNAN NAIR
Chief, Cataloging Branch

FL-171
Jul 93

DISTRIBUTION STATEMENT A:

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

DISTRIBUTION STATEMENT B:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES ONLY; (Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT C:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND THEIR CONTRACTORS; (Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT D:

DISTRIBUTION AUTHORIZED TO DOD AND U.S. DOD CONTRACTORS ONLY; (Indicate Reason and Date Below). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT E:

DISTRIBUTION AUTHORIZED TO DOD COMPONENTS ONLY; (Indicate Reason and Date Below). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT F:

FURTHER DISSEMINATION ONLY AS DIRECTED BY (Indicate Controlling DoD Office and Date Below) or HIGHER DOD AUTHORITY.

DISTRIBUTION STATEMENT X:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND PRIVATE INDIVIDUALS OR ENTERPRISES ELIGIBLE TO OBTAIN EXPORT-CONTROLLED TECHNICAL DATA IN ACCORDANCE WITH DOD DIRECTIVE 5230.25, WITHHOLDING OF UNCLASSIFIED TECHNICAL DATA FROM PUBLIC DISCLOSURE, 6 Nov 1984 (Indicate date of determination). CONTROLLING DOD OFFICE IS (Indicate Controlling DoD Office).

The cited documents has been reviewed by competent authority and the following distribution statement is hereby authorized.

A

(Statement)

OFFICE OF NAVAL RESEARCH
CORPORATE PROGRAMS DIVISION
ONR 353
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

(Controlling DoD Office Name)

(Reason)

(Controlling DoD Office Address, City, State, Zip)

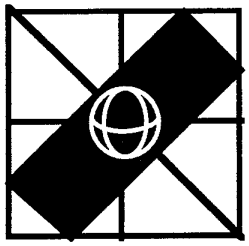
DEBRA T. HUGHES
DEPUTY DIRECTOR
CORPORATE PROGRAMS OFFICE

19 SEP 1995

(Signature & Typed Name)

(Assigning Office)

(Date Statement Assigned)



Proceedings of the

IFIP WG 11.3



Eighth Annual WORKING CONFERENCE on DATABASE SECURITY

Bad Salzdetfurth, Germany 23 - 26 August 1994

sponsored by

IFIP Working Group 11.3 **University of Hildesheim**

Editors

J. Biskup, M. Morgenstern, C. Landwehr

DTIC QUALITY INSPECTED 4

ISSN 0941-3014 Hildesheimer Informatik-Berichte 20/94, August 1994,
Universität Hildesheim, Institut für Informatik, 31141 Hildesheim, Germany

Preface

These proceedings contain the papers presented at the Eighth Annual IFIP WG 11.3 Working Conference on Database Security held in Bad Salzdetfurth near Hildesheim, Germany, August 23-26, 1994. The conference has been sponsored jointly by the International Federation for Information Processsing (IFIP) Working Group 11.3 on Database Security and the University of Hildesheim. Furthermore the U.S. Office of Naval Research supported travel of many U.S. participants. As with its predecessors the purpose of the conference was to present original work in database security research and practice, to enable participants to benefit from personal scientific discussions and to expand their knowledge, to support the activities of the Working Group, and to disseminate its results.

All submitted papers were reviewed by working group members and a small number of external reviewers. Based on these evaluations 18 papers were finally accepted for presentation. In addition the program included a session on status reports on current projects and a panel discussion on perspectives on database security. These sessions are also documented in these proceedings by short abstracts.

We highly appreciated two invited lectures that are also included in these proceedings. Ab Bakker from BAZIS, Leiden, Netherlands, presents his deep insight into security in health care systems, and thereby we were able to continue the discussion of security in this important field of application. Klaus Dittrich from University of Zürich, Switzerland, challenges the security community with current trends in database technology to which he himself has substantially contributed.

A Working Conference is based on the joint efforts of many people. We thank all of them sincerely: the authors of submitted papers, the reviewers, the participants, in particular the invited speakers. We are also particularly grateful to Jimmy Brüggemann and Christian Eckert for local organization.

Program Co-Chair

Joachim Biskup
Institut für Informatik
Universität Hildesheim
Samelsonplatz 1
Postfach 101363
D-31113 Hildesheim
+49(5121)883-731 (voice)
+49(5121)883-732 (fax)
biskup@informatik.uni-hildesheim.de

Program Co-Chair

Matthew Morgenstern
Design Research Institute
Cornell University
5144 Upson Hall
Ithaca, NY 14853
U.S.A.
+1(607)255-9899 (voice)
+1(607)254-4742 (fax)
morgenstern@cs.cornell.edu

IFIP WG11.3 Chair

Carl E. Landwehr
Naval Research Laboratory
Code 5542
4555 Overlook Ave., SW
Washington, DC 20375-5000
U.S.A.
+1(202)767-3381 (voice)
+1(202)404-7942 (fax)
landwehr@itd.nrl.navy.mil

Reviewers for IFIP WG 11.3 Working Conference on Database Security, 1994

Bertino, Elisa

Biskup, Joachim

Brüggemann, H.H.

Burns, Rae K.

Ciampichetti, Alessandro

Costich, Oliver

Cox, Lawrence H.

Cuppens, Frederic

Dittrich, Klaus R.

Eckert, Christian

Fernandez, E.B.

Gollmann, Dieter

Gudes, Ehud

Hosmer, Hilary H.

Hu, Mei-Yu

Jajodia, Sushil

Jonscher, Dirk

Kang, Myong H.

Kang, Iwen

Keefe, Thomas F

Landwehr, Carl E.

Lin, T. Y.

Lunt, Teresa F.

MacEwen, Glenn

Martella, Giancarlo

McDermott, John

Michael, Bret

Morgenstern, Matthew

Notargiacomo, LouAnna

Pal, Shankar

Pernul, Guenther

Pfitzmann, Birgit

Qian, Xiaolei

Samarati, Pierangela

Sandhu, Ravi

Schaefer, Marvin

Sibley, Edgar H.

Smith, Gary W.

Smith, Kenneth

Steinke, Gerhard

Thomas, Roshan

Thuraisingham, Bhavani

Varadharajan, Vijay

Warner, Andrew C.

Wiseman, Simon

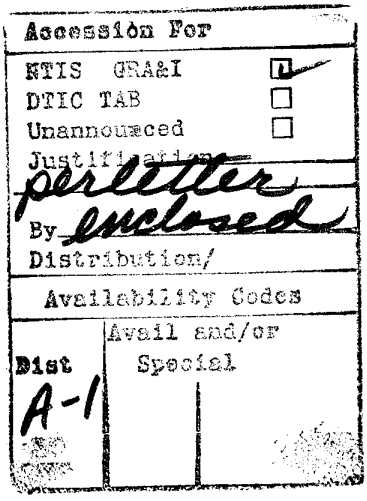


Table of Contents

Preface .....

Invited Lectures

Special care needed for the heart of medical information systems
A. Bakker .....

Current trends in database technology and their impact on security concepts
Klaus Dittrich .....

User groups and roles

Access rights administration in role-based security systems
M. Nyanchama / S. Osborn .....

User group structures in object-oriented database authorization
E.B. Fernandez / J. Wu / M.H. Fernandez .....

User-role based security in the ADAM object-oriented design and analyses environment
M.-Y. Hu / S.A. Demurjian / T.C. Ting .....

Database architecture

The SINTRA data model: structure and operations
O. Costich / M.H. Kang / J.N. Froscher .....

The b2/c3 problem: how big buffers overcome covert channel cynicism in trusted database systems
J. McDermott .....

Trusted RUBIX: a multilevel secure client-server dbms
J.P. O'Connor .....

Inference analysis and control

A practical formalism for imprecise inference control
J. Hale / J. Threet / S. Shenoi .....

Hypersemantic data modeling for inference analysis
D.G. Marks / L.J. Binns / B.M. Thuraisingham .....

Database models

A multilevel secure federated database
M.S. Olivier .....

A new authorization model for object-oriented databases
E. Bertino / F. Origgi / P. Samarati .....

The integration of security and integrity constraints in MOKUM
R. P. van de Riet / J. Beukering .....

Queries, updates, and transactions

Field level classification and SQL
S. Wiseman .....

Degrees of isolation, concurrency control protocols, and commit protocols
V Atluri / E. Bertino / S. Jajodia .....

Status reports on current projects

Trusted Ontos prototype
M. Schaefer / S.A. Wade .....

Panel: Perspectives on database security

Panel chair: M. Morgenstern .....
Panelists: J. Biskup, K. Dittrich, C. Landwehr, M. Schaefer

Policy modelling

Providing consistent views in a polyinstantiated database
L. Cholvy / F. Cuppens .....

Secure logic databases allowed to reveal indefinite information on secrets
A. Spalka .....

Access control and application design

A fine grained access control model for object-oriented dbms
A. Rosenthal / J. Williams / W. Herndon / B. Thursaisingham .....

Creating abstract discretionary modification policies with reconfigurable data objects
T. Gross .....

Security guidelines for database systems development
G. Pangalos .....

Invited Lectures

SPECIAL CARE NEEDED FOR THE HEART OF MEDICAL INFORMATION SYSTEMS

ALBERT R. BAKKER
BAZIS FOUNDATION
SCHIPHOLWEG 97 2316XA LEIDEN, THE NETHERLANDS

abstract

First characteristics of medical information systems are described. Next the role of the database is considered in more detail, the resulting requirements for security of medical databases are considered.

Special attention is given to the requirements resulting from use of the database for medical audit. It is concluded that in databases holding patient care data strict time-stamping is required. Even after expiration of the specified storage period there has to be a certain period when the data should be dormant, in that state they are accessible by the auditor only.

1. INTRODUCTION

Data play an important role in modern health care. Data are recorded for several purposes:
- supporting the own memory of the health care professional, to allow him to do his diagnostic and therapeutic work,
- as a communication vehicle between members of the team that is caring for the patient,
- as a communication vehicle with medical support departments, like laboratories, radiology, pharmacy, etc.,
- as input to the financial administration, the reimbursement, the budgetting process and the management of the care facility,
- as potential input for (epidemiological) research and education.

With the growth of medical knowledge and medical technology the amount and diversity of data to be recorded is increasing rapidly. Because also the need for medical specialisation is increasing the need for communication with other specialists (also those within specialised support departments) is increasing. This lead to a situation where it is estimated that handling of information in a broad sense (including images) generates about 30% of the costs of a hospital.

For the provision of health care we find a wide range of institutions and facilities (hospital, general practitioner, pharmacy, community care, fysiotherapist, etc). Until now the amount of data exchanged between these different health care providing organisations/persons is limited, the exchange of information is now primarily within the walls of the institution. Both the emerging technical facilities for communication (WANs) and the need to improve the efficiency in health care will lead to more emphasis on external communication.

Medical data about patients often are of a rather sensitive nature and deserve protection, the "medical secret" is a worldwide principle and mentioned explicitly in the professional oath (Hyppocratic oath) [1]. As long as medical care was supplied in a direct one-to-one patient-doctor relation the application of the medical secret was almost unambiguous.

However, in modern health care many persons are involved like: other members of the care team (doctors as well as nurses), secretaries, laboratory technicians, medical records clerks, administrators. In addition to that, insurance companies ask for data as well as researchers. The practical rules for access to patient data differ between countries and between institutions albeit that the general principles as described in the Council of Europe's Regulations for Automated Medical Databanks R(81) [2] are rather well accepted and are the base for most data protection laws.

2. CHARACTERISTICS OF MEDICAL INFORMATION SYSTEMS

When security is concerned we find in general special attention for medical information systems. In this respect can be mentioned ; the recommendation of the Council of Europe and special clauses in many national laws and regulations referring to medical information systems. Furthermore we find dedicated committees (e.g. CEN TC251 wg 6; IMIA wg4; EFMI wg2), and dedicated (working) conferences [3],[4],[5],[6],[7].

What are the reasons for such special attention? Are there special characteristics of medical information systems that justify this special attention? In this paragraph this question is addressed as background for the security aspects of medical databases. A general overview of security aspects of medical information systems can be found in [8].

Although we find information systems in almost any type of health care organisation we will focus here on the most complex example of medical information systems, the systems that support large hospitals. Such hospital information systems (HIS) offer a wide range of functions to a wide range of users (of different disciplines) [9]. Because of the interrelation of activities in the hospital integration is a key characteristic of successful hospital information systems. At present the functions of such systems are, apart from some EDI facilities to support communication with the outside world, serving the internal information processes within the institution.

A central point of reference in such system is the patient. Data about his health condition and treatment are needed by many workers in the hospital. The data have to be available at the workplace, at the moment they are needed, in a presentation geared towards the needs of the specific user. However, only those data have to be presented that the user is allowed to see ("need to know principle").

The data are stored in a database. The collection of data of one patient is often referred to as his "electronic medical record", to indicate that the data are gradually becoming the electronic equivalent of the paper file holding the patient data. This electronic medical record until now covers only alpha-numeric data. Images are only stored in information systems in seldom cases, e.g. in Picture Archiving and communication Systems (PACS) [10]. In this paper special security requirements for PACS will not be discussed.

Data in a HIS are intended to play a role in the care process. This process goes on continuously, at least for inpatients, so, by consequence the functions of the system and the data about the patient have to be available round-the-clock.

Because the data presented by the system play a direct role in the care process their integrity is important. Wrong data presented may harm the patient directly because they may trigger wrong actions.

Although the principles for access rights to patient data are the same in most institutions, the translation into rules to be applied for checking these access rights may differ significantly between institutions for internal access. For access of external users the differences are often even larger. The question whether the patient has the right of access to his own data is answered differently in different countries. In some countries the answer is a clear unconditional "yes" while in other countries only access is allowed through a physician as an intermediary.

Within an institution the right of access is not only determined by the position/profession of the user, but also by the answer to the question whether he is involved in the care process and if so in what role. So a physician will have the right to read diagnoses recorded in the system, but only for "his patients". Whether a patient is "his patient" is determined by looking at logistic patient data (admission data, appoinment data, waiting lists). These data being recorded within the database itself.

For access to patient data in case of medical emergencies special provisions have to be made. In such situations it is often impossible to check automatically whether the health care professional has a care relation with the patient. A way out can be found by assigning some users (doctors and nurses) the special authority to bypass (in emergency situations) the check on patient relation. When the system refuses access because no such relation can be detected they can state that the request is dealing with an emergency (supported by an input message of a minimum length explaining the reason for the request) after which the requested data are supplied. Of this violation of the protection a message is sent by the system to a supervising authority (e.g. the head of the medical records department, or the medical director) who checks whether the emergency really occurred.

Sometimes it is attractive from a perspective of efficiency that several hospitals share the same computer centre and run the same hospital information system. As long as the databases and the software are kept strictly separated this does not lead to complications. However when the hardware facilities are shared there will be a strong pressure to share also certain categories of data like: general identification data, patient identifier, patient insurance status. As soon as such sharing is made possible there will be a demand to share more data, e.g. results of laboratory tests. Such sharing of data may complicate the database management.

The requirements for access to the HIS and its availability are high. Nowadays for a mature HIS the number of terminals or workstations exceeds the number of beds of the hospital. The availability should be at least 99.7% round-the-clock.

3. THE MEDICAL DATABASE

In the database different categories of data will be stored:

- patient data, both medical, administrative and logistic data. Examples of medical data are: results of laboratory tests, discharge letters, diagnoses, vital signs, diets, etc. Examples of administrative data are: insurance data, (pending) invoices;
- data on the resources of the institution and their utilisation, e.g. personnel data, budgets, stocks, bed occupancy, etc;
- logistic data; waiting lists, appointment schedules, reservations, etc
- reference data, like diagnosis codes, list of drugs, general practitioners, treatment protocols, etc.

Although the variety of data is large and the volume of transactions will be high, the database can in general logically be considered as relational, although often for efficiency reasons a special structure is chosen. For most records representing medical facts multi-occurencies are to be expected, e.g. the result of an ECG may occur several times in the medical record of a patient.

When several institutions decide to share the computer facilities the implementation of several incarnations of the system on the same computer in principle does not lead to special requirements. However, the demand will arise to flexibly share data, this will lead to special requirements for the database management system.

Some quantitative data on the database of a typical European university hospital (1000 acute beds, 300,000 outpatient visits per year) are given here as an illustration:
- number of patients registered > 800,000
- number of diagnoses recorded >500,000
- number of radiology reports > 500,000
- number of lab test results > 2,000,000
- number of terminals/workstations > 1,000
- number of database actions (read, write, delete, update) in the dayshift 10 - 15 M per day, of which 3% additions
- total volume 6 Gbytes
- number of tables in database 6,000
- number of different attributes 20,000.

4. SECURITY REQUIREMENTS FOR MEDICAL DATABASES

The requirements for security are considered here for the three CIA effects: Confidentiality, Integrity, Availability.

Confidentiality requirements

The facilities for access control should be refined and flexible. Refinement because of the wide variety of data and the wide range of roles of users. Flexible to allow each institution to map its rules for access on the facilities of the database management system. Access control should be based on:
- access patterns for the different user roles to be distinguished,
- when access is requested to patient data, the existence of a care-relation of the user with the patient concerned,
- the existence of an emergency situation,
- the entity accessed, and sometimes the specific attributes of that entity.

A special point of concern are queries that retrieve data from the database, not directly through patient identification. Such search questions may nevertheless reveal sensitive information, even if only the number of occurrences of specific situations are reported.

Integrity requirements

Because data from medical databases play a role in the direct care process already now, and such use can be expected to increase rapidly, special attention is needed for the integrity of the data. Incorrect data might lead to wrong diagnostic or therapeutic decisions, causing damage to the health of the patient. Measures have to be taken to:

- avoid loss or destruction of data;
- unauthorized modifications in the programs;
- check data at entry on consistency and plausibility (taking into consideration other data stored already (irrespective of the access rights of the user);
- avoid inconsistent presentation of data because of limited access rights.

The latter point is explored here slightly further. Polyinstantiation is not unusual for some categories of data, e.g. results of laboratory tests. For other categories it is highly undesirable (e.g. bloodgroup). It should be specified for which categories such polyinstantiation is allowed. Undesirable polyinstantiation could be avoided by applying the rule that if someone is not allowed to read a certain record type he is not allowed to create it either.

As a typical integrity problem in medical databases we consider here the situation where data of a patient are stored under two different patient identifications, that lateron turn out to be dealing with the same patient. Such a situation may e.g. occur in emergency admissions or when samples for laboratory tests are offered with a limited amount of patient identification data. If it can not be made sure that the patient is the same as one already recorded in the databank, the only option is to consider him as a new patient. Lateron it may turn out that the patient was already in the database and the medical (and administrative) records have to be merged. This merging may lead to conflicts of consistency, that would normally have been handled in an interactive way at the moment of data entry. Now special provisions are necessary to cope with such inconsistencies by e.g. the medical records officer.

Such consistency problems will occur on a large scale if two institution would merge and by consequence would like to merge their databases. The problem will also deserve attention as soon as two institutions in a region decide to share some categories of data.

Availability requirements

When medical information systems play an important role in the care process, as is for instance the case for most hospital information systems, there is a need for high availability. Typical requirements are availability > 99.7% round-the-clock. Apart from the availability percentage also the time needed to resume operations after an interruption is important. Three situations can be distinguished:

- simple restart possible, restoring some system parameters

and checking vital system data; this typically should not take more than 10 minutes;
- database damaged (either by hardware malfunction, software problem or human error); recover necessary from safe-copies plus logged mutations; this should not take more than 4-6 hours;
- computer centre (or most of the equipment located there) destroyed by a disaster, e.g because of a large fire; in this situation external back-up facilities will be necessary, e.g. a mobile or a remote back-up computer centre. In the latter case data communication facilities have to be available.
Although such disasters will only occur very seldom (less than once in 50 years) the interruption of services in the hospital should be no longer than 24 hours.

With the increasing role of the information systems in direct patient care the requirements will become stricter. Mirroring of the database is one of the techniques to reduce the risk for loss of the database followed by a time-consuming recover action. So-called "non-stop operation" should be considered seriously. However, it should be realized that this provides no protection against disasters, human failures and incorrect new versions of software subsystems. Anyhow non-stop facilities fall beyond the database software as such.

5. MEDICAL AUDIT ASPECTS

Although the introduction of information systems in the direct patient care has proceeded slower than expected, we see nowadays that they are at several places replacing parts of the functions of the paper medical record.

Medical audit tries to answer the question whether a health care professional acted in a specific situation in a responsible way in view of what he knew or ought to have known. This both refers to professional knowledge and to patient data of the case. The interest in medical audit can be expected to increase in view of the tendency to more often raise the issue of liability when the result of the medical treatment is not what the patient expected. If an information system is available to the health care professional he can be expected to use it in his work. Responsible behaviour will also comprise use of the data from the system.

This implies that in an audit procedure the auditor needs to have the possibility to see what data (on a patient) a certain health care professional could have seen at a certain moment. Although it is easily recognized that such a requirement makes sense, there are no documented examples of its fulfilment in operational systems. Let us consider the implications:
- all data in the database have to be time-stamped, to be able to select those data that were available at a specified moment;
- after commands for deletion or modification of data, the old values have to be preserved in a way that they are not shown in routine operation of the system, but are accessible for the audit process;
- this even applies for data that have to be deleted after expiration of the specified storage period; at least as long as an audit should still be possible. This requires a reformu-

lation of the widely accepted principle in data protection regulations that for the various types of data stored the maximum storage period has to be specified after which they must be destroyed;
- there has to be an audit mode, offering the auditor the possibility to masquerade as a certain health care professional (taking on his rights) and giving as a result to requests to the database the answers as they would have been at a specified moment (in the past). This implies that also the history of those data that play a role in checking access rights should be stored, which is not always current practice

Fulfilling these requirements will be far from trivial, it would require a complete overhaul of existing medical information systems. It might be considered to use as a vehicle for the audit process the safe-copies of the database that are made daily. This would still require an audit mode (for masquerading the health care professional), but would avoid the strict time-stamping, the maintaining of historical data and the dormant mode of deactivated data. However, medical audit will often have to deal with critical health situations of patients, in those situations the contents of the electronic medical record will change rapidly. A snapshot frequency of once per day is most probably far too low. This is underpinned by the average length of stay of patients that is around 10 days or less in most hospitals in the western world.

It should be emphasized that the requirements of medical audit as formulated here have not yet been raised by the medical auditors, however it seems highly probable that we can expect them in a few years. It is better not to wait until the problem arises, the various disciplines involved should consider now what the requirements are and how these can be met by the technology. Use of daily safe-copies may be an interim solution while a more fundamental solution is prepared.

6. CONCLUSION

In this paper database security for medical information systems has been considered, especially for hospital information systems. Such systems are found to be very complex, the wide variety of data and users on one side, and on the other side the types of the applications make them an interesting case for security considerations.

Already now such systems play a role in the direct patient care, such role can be expected to increase rapidly. This will have as consequence that the facilities of the system will be taken into consideration in medical audit. The question posed being:"did the health care professional act in a responsible manner, in view of the information that was available to him ?" Because the information system is an important source of information, it will be necessary to be able to replay the patient information that at a certain point in time would have been available for the health care professional concerned. The consequences of fulfilling such requirement seem to be dramatic, solutions to cope with this challenge deserve attention.

7. REFERENCES

[1] Roger France FH, Gaunt PN. The need for security, a clini-

cal view. International Journal of Bio-Medical Computing 35 (Suppl I),1994. pp 189-194.

[2] Council of Europe Regulations for Automated Medical Data Banks. Recommendations R(81), Strasbourg,1981.

[3] Griesser GG,ed. Realization of Data Protection in Health Information Systems. Amsterdam: North-Holland Publ Comp, 1977.

[4] Griesser GG, Bakker A, Danielsson , et al ,eds. Data Protection in Health Information Systems, Considerations and Guidelines. Amsterdam: North-Holland Publ Comp, 1980.

[5] Griesser GG, Jardel JP, Kenny DJ, Sauter K, eds. Data Protection in Health Information Systems, Where do we Stand ? Amsterdam: North-Holland Publ Comp, 1983.

[6] The Commission of the European Communities DG XIII/F AIM. Data Protection and Confidentiality in Health Informatics. Amsterdam: IOS Press, 1991.

[7] International Journal of Bio-Medical Computing 35 (Suppl I), 1994.

[8] Bakker AR. Security in Medical Information Systems. Yearbook of Medical Informatics '93. Schattauer, Stuttgart,1993.

[9] Bakker AR, Bryant JR, Ehlers CTh, Hammond WE., eds. Hospital Information Systems; Scope-Design-Architecture. Amsterdam Elsevier Science Publ, 1991.

[10] Huang HK, Ratib O, Bakker AR, Witte G. Picture Archiving and Communication Systems (PACS) in Medicine. Berlin. Springer Verlag, 1991.

[11] Bakker AR. Presentation of electronic patient data and medical audit. International Journal of Bio-Medical Computing 35 (Suppl I),1994 pp 65-69.

Current Trends in Database Technology and Their Impact on Security Concepts

— Summary —

Klaus R. Dittrich

Institut für Informatik der Universität Zürich
Winterthurerstrasse 190, CH-8057 Zürich
dittrich@ifi.unizh.ch

Since about a decade, a large part of research and development in the area of database management systems (DBMS) is being devoted to the issue of extending their functionality. The general goal behind these efforts is to efficiently provide advanced data and information management services within standard software, thus avoiding the need for their repeated design and incorporation within individual application programs. The basis of all such approaches is – compared to traditional (e.g. relational) systems – a more comprehensive and precise representation of real world semantics within the database itself. In consequence, new generation DBMS are about to make database technology amenable to a much broader range of applications than traditional products do.

From the viewpoint of security, advanced DBMS concepts present both, new opportunities and new challenges. We will look at three prominent examples, namely object-oriented, active, and federated DBMS, and briefly sketch their impact on some security issues, primarily (discretionary) access control. Furthermore, we will touch the important issues of DBMS construction and security design.

Object-oriented DBMS

Object-oriented DBMS support an object-oriented data model, i.e. a data model based on the notions of objects with values, definable behavior and identity, encapsulation, classes, and class inheritance. Compared with the relational model, object-orientation allows for a much more extensive and accurate modeling of real world information. This is due to the support of complex values (using constructors like tuple, set, list, etc., which may be applied recursively), object structures (to express various foms of object associations), and user-defined operations on each class (to express behavior beyond the simple retrieval and update of data).

As access control for databases obviously has to reflect the data units handled by the respective data model, known database access control features at least have to be adjusted to the notion of "object". It shows that this is not that easy as might seem at first glance, particularly due to object structures and their variety of semantics. On the other hand, if we can model the real world more precisely thanks to object-orientation, we should also be able to fine-tune access control rules much better than otherwise. In particular, we can now much more exactly differentiate between the object operations to be allowed or denied for individual users, as those may carry much more real world semantics than mere "read", "write" and similar ones.

Active DBMS

Active DBMS allow – beyond providing all regular DBMS-features – the recognition of user-defined situations in the database and beyond, and the execution of user-defined reactions when such a situation occurs. The popular specification paradigm for active DBMS are so-called event/condition/action rules (ECA-rules)

on <event> **if** <condition> **do** <action>

which can be defined in addition to the regular database schema. When an event is detected by the system, the condition is checked on the database and if it holds, the specified action is executed. Many details have to be considered in such systems, including e.g. the kinds of events, conditions and events that are supported and the execution of rules within the given transaction model. In particular, the power of active DBMS is highly dependent on the event model which may include the occurrence of database operations, but also externally raised events, time events, and various combinations thereof.

Obviously, active rules and their execution have to be subject to security, too. In this respect, a rule and its constituent parts can be regarded as database elements, but will most probably still require particular treatment with regard to access control. Furthermore, it turns out that subtle points may arise as to which access rights have to be applied when a reaction is executed as a consequence of an event triggered by some other action (running on behalf of a particular user). On the other hand, ECA-rules also allow for the flexible and dynamic specification of rather sophisticated security policies, either by direct use or by acting as a target mechanism for some sort of security specification language.

Federated DBMS

A federated DBMS provides for the interoperation of (probably heterogeneous) component DBMS under one "common roof". Such federations are mainly conceived to facilitate the integration of existing "information islands", but may also help to allow the introduction of e.g. an object-oriented DBMS in an environment where a more traditional DBMS is already in use (affectionately called "legacy systems") and where both kinds of system have to interoperate. In this case, the autonomy of component systems is an important issue.

Once again, there are two sides of the coin as far as security is concerned. Whereas especially component autonomy raises various problems when it comes to authorization and access control, we can – under a number of assumptions – even hope to retrofit more elaborate security features to "poor" component DBMS provided they are operated through the federal layer of the system.

DBMS construction

Given the increasing number and complexity of DBMS that may be required, the DBMS community gets more and more interested in how to efficiently build DBMS, without having to start from scratch all the time. Obviously, this means to apply

state-of-the-art software engineering principles – an issue that by the way has been largely neglected in the past. As a result, extensible DBMS and DBMS construction systems exploiting configuration and generation techniques have been suggested.

In particular, it seems to be promising to regard a DBMS as a collection of resource managers that cooperate under the control of some sort of "broker". In such a view, it has to be determined where various security mechanisms fit in to meet all requirements. Though work in this direction is still in its infancy, it can be expected that some basic security functionality will be required in the broker, while most others can be nicely organized into a variety of "security managers".

Security design

Allowing for more comprehensive modeling of the real world by means of data models, active rules etc. will improve the quality and maintainability of software systems and also improve the efficiency of the software development process. However, as real world systems we want to automate are usually rather complex, the use of advanced modeling facilities is unfortunately complex, too. In consequence, it is also often everything but straightforward to apply advanced security features in the appropriate way. As a consequence, it is not sufficient (though very important!) to provide the technical means for powerful and effective security mechanisms. In addition, security administrators have to be helped in their job by appropriate design methodologies and tools which allow them to formulate their requirements and map these systematically to the relevant mechanisms.

In summary, current trends in database technology indeed do have considerable impact on security concepts, in terms of both, better solutions that can be supported and new problems that need to be solved. Unfortunately, commercial products in this area are – once again! – very slow to incorporate security features that are as advanced as the rest of the system from the very beginning. At best, they are going to retrofit them to the system in later releases. The security community is thus challenged not only to device and evaluate appropriate concepts, but also to push for and foster the necessary technology transfer to DBMS builders and users.

User groups and roles:

Chair: D. Spooner

Rensselaer Polytec. Inst., NY

Access Rights Administration in Role-Based Security Systems [1]

Matunda Nyanchama & Sylvia Osborn
The Department of Computer Science
The University of Western Ontario
London Ontario N6A 5B7 Canada
FAX: (519) 661-3515
email:{matunda,sylvia}@csd.uwo.ca

Abstract

This paper examines the concept of role-based protection and, in particular, role organization. From basic role relationships, a model for role organization is developed. The role graph model, its operator semantics based on graph theory and algorithms for role administration are proposed. The role graph model, in our view, presents a very generalized form of role organization for access rights administration. It is shown how the model simulates other organizational structures such as hierarchies [TDH92] and privilege graphs [Bal90].

Keywords: Roles, role-based protection, access control, privilege graph, least privilege, Role Graph.

1 Introduction

Role-based protection is a flexible means of administering large numbers of system privileges especially for large databases. A privilege is a *unit* of access to system information. A role is a named collection of such privileges [Bal90, KM92, NO93b]. User authorization to a role grants the user access to the privileges defined in the role.

The advantage of role-based protection is that it eases the administration of privileges because of the flexibility with which roles may be configured and reconfigured [TDH92, NO93b]. System security is served further when the role configuration process is based on the principle of *least privilege* in which a role is equipped only with sufficient privileges to facilitate the intended duty requirements [Tho91].

In an organization with a large number of diverse duty requirements, the number of roles can proliferate as new roles are defined to meet specific duty requirements. Some roles can have overlapping functions (hence overlapping privileges) while others need not overlap. The need to have some formal manner of tracking the distribution and administration of privileges is important to ensure proper exercise of both responsibility and system security. It is important to have a means of formally expressing role relationships – one which reflects the manner of distribution of privileges in a system.

This paper examines what we consider *basic relationships* that can exist among roles in an organization and their application in modeling role organization. Using these basic relationships as the foundation, a model for role organization is proposed. It is possible for the privilege sets of two roles to completely overlap (one is a subset of the other), partially overlap (have a common subset) or have a common superset. These relationships, along with the concepts of *maximum* and *minimum* privilege sets form the basis of the role graph model. To demonstrate the expressive power of this model, we illustrate how it simulates organizational structures such as hierarchies [TDH92] and privilege graphs [Bal90].

[1] To Appear in Database Security VIII: Status & Prospects, August, 1994.

In the next section we discuss the concepts of privileges, roles and the advantages of role-based protection. We formally define the term *role*, as used in this paper, and motivate the need for formal role organization. In section 3 we discuss the basic relationships that can exist among roles and introduce operators to model these relationships. We regard these relationships as forming the basis of role organization modeling. Section 4 formally presents the role graph model, and gives algorithms for role administration. Section 5 discusses model simulation of other role organizational structures. Section 6 contains the summary and conclusions.

2 Introduction to Roles

2.1 Basic Definitions

The idea of a role arises out of the need to provide duty functionality which is then authorized as a single unit. A role can be seen as a job, office, set of actions of a role-holder, a collection of responsibilities and functions or a collection of privileges pertaining to some duty requirements [DM89, Bal90]. A role exists as an entity separate from the role holder or role administrator. It should be equipped with sufficient functionality to enable an authorized user to achieve the duty requirements associated with the role. Hence a clerical role will be given sufficient access rights to enable an authorized user, or user group, to perform clerical duties. Baldwin [Bal90] terms these **Named Protection Domains** (NPDs). Such a role specification captures the responsibilities, rights and obligations associated with what Dobson and McDermid [DM89] term a *functional role*.

The other important component of role definition is its *structural* [DM89] aspect which captures a role's relationship with other roles. For purposes of this paper, we shall use the term role to refer to the functional aspect while the structural aspect of role relationships will be captured by the structure defining their relationships–in our case a *role graph* model.

A role is defined in terms of privileges. A privilege, on the other hand, is defined in terms of access modes and can be viewed as a unit of access rights administration.

Definition 1 *Privilege: A privilege is a pair (x, m) where x refers to an object and m is a non-empty set of access modes for x.* □

The object referred to by x can be a protected data item, an object-oriented (O-O) class definition or extent, a complex object, a resource (e.g. printer), etc. x can be any name or identifier which uniquely specifies the associated object. m, the set of access modes, is composed of valid modes of access to x. Its specification and administration can be subjected to a range of security policies. In systems with simple access modes such as reads, writes, executes, etc. m, is a subset of these access modes. In complex systems, these access modes can be composed of a series of or nested applications of reads, writes and executes. Where x is an object in an O-O environment, m would be the execute mode of one or more methods. In transactional systems, m would be a list of transactions that facilitate access to x. The exact nature of x and m is a matter of the application environment and the associated security policy [NO93a]. Since privileges are intended for security administration, the security policy must specify how they are administered. In our case, the initialization and modification of a privilege must be authorized.

Definition 2 *Role: A role is a named set of privileges. It is a pair (rname, rpset) where rname is the role name and rpset is the privilege set.* □

A role's name **rname** uniquely identifies a role in a system. We use dot notation to refer to a role's name and privilege set. Thus for a given role r, r.rname and r.rpset refer

to the name of the role and its privilege set, respectively. Let $\mathcal{PV}$ denote the universal set of privileges in a given system, and $\mathcal{R}$ the universal set of roles. We also define a function $\Psi : \mathcal{R} \mapsto \mathcal{PV}$, which enumerates the privileges of a given role, so that for every $r \in \mathcal{R}$, $\Psi(r) = \{pv_1, \cdots, pv_n\} = r.rpset$.

2.2 Strengths of Role-Based Protection

Role-based protection offers *flexibility* in system privilege administration [TDH92, NO93b]. User access rights can be varied either by *explicit* authorization (or revocation of authorization) of a user to a role or by indirectly varying the role privilege set. Further advantage is gained if users are organized into groups such that authorizations are given to groups, as opposed to individuals.

Given that system privileges can be very fine-grained, roles offer a means of managing them incrementally. Considering the manner in which privileges can be assigned/revoked to/from a given role, this method approaches a continuum in system privilege administration [NO93b]. A related advantage is that role-based protection can be used to enforce the principle of *least privilege* where a role is defined to have only the necessary functionality required for the associated duties [Tho91].

This approach offers a simplification of the complexity of system privilege management. With a suitable organizational framework capturing role relationships, it is possible to analyze the implications of given authorizations. Moreover, such a formal framework lends itself to the development of analytical tools. It is also possible that management tools for access rights administration can be used in role management.

Given that role-based protection is designed with a given application in mind, this method provides a chance for incorporation of application level security constraints and semantics [Tho91]. An associated advantage is that roles allow for *multidirectional* information flow policies [Tho91] unlike such models as Denning's role graph model [Den76] and Bell and LaPadula's [BL75] multilevel model. As well, unlike these traditional models which specify what information flows *should not* take place, role-based protection affirms which information flows *can* take place [GMP92].

2.3 Roles & Access Rights Administration

Roles act as *gateways* to system information. The privilege set of a given role determines what information is available via the role. One advantage of role-based protection mentioned in the previous section is that access to system information is accomplished at two levels: via explicit authorization to a role or via inclusion of some privilege in a role. We term the former *user-role* authorization while the latter is termed *role-privilege* authorization (see figure 1). A third form of authorization is *role-role* authorization [Bal90] in which one role is authorized another's privileges. We address each of these in turn.

In **user-role** authorization, a user/group is authorized access to system privileges available via the role. Such authorization must be specified in a role's access control list. For each role, such an access control list contains the user identifier for each user authorized to the role.

Let $\mathcal{UID}$ be the set of all user identifiers, and $\mathcal{GID}$ the set of all group identifiers; $\mathcal{ID} = \mathcal{UID} \cup \mathcal{GID}$.

Definition 3 Access Control List: *A role access control list (racl) is of the form:* $[id_1, \cdots, id_n]$, *where* $id_i \in ID$. □

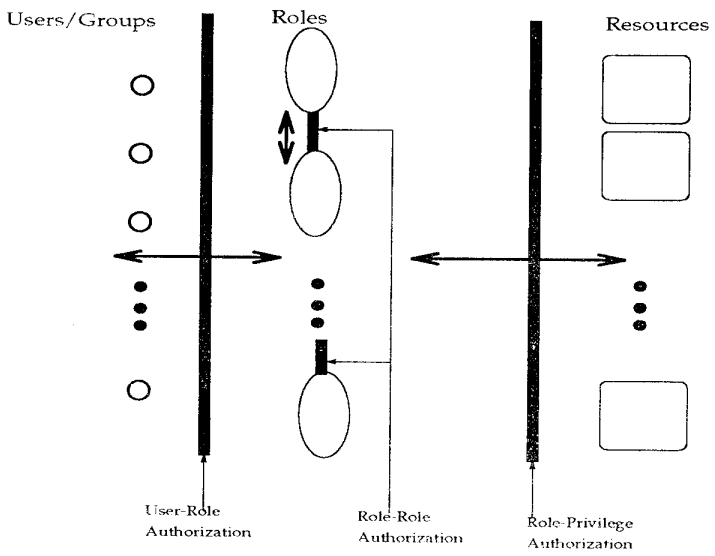


Figure 1: Three Kinds of Authorizations

In a secure system all roles must have access control lists, i.e. $\forall r \in \mathcal{R}, \exists$ r.racl $= [\cdots, id_i, \cdots]$. A role with an associated access control list is called a *secure role.*

Definition 4 *Secure Role: A secure role is a named collection of privileges along with its access control list. It is a triple (rname,rpset,racl), where rname is the role name, rpset is its privilege set and racl is its access control list.* □

Role-privilege authorization involves role configuration in which a privilege is added to the role's privilege set. **Role-role** authorization [Bal90] forms the third kind of authorization. If a role A is authorized to access a role B, it means that all of B's access rights are available via role A. In other words, B's privileges are a proper subset of the privileges of A. Role-role authorization is an aspect of role structure.

Example 1 Suppose we have two roles: **clerk** and **supervisor** in which the **supervisor** role has a role authorization to the **clerk** role. This means that the clerk's access rights are available to the supervisor. A user authorized to the **supervisor** role can perform whatever a user authorized to the clerk role can do.[2] We can view the privilege relationships between the two roles as $\Psi(clerk) \subseteq \Psi(supervisor)$. □

This paper examines role-role authorizations which define role relationships. These have implications on role organization and access rights administration. Role-role authorizations can be complex. To capture the role-relationships completely and be able to carry out an analysis of the implications of privilege assignment and distribution in a system can be very complex without some formal organizational structure. Complexity of analysis of system privilege distribution is one short-coming of role-based protection [TDH92, NO93b].

Baldwin's approach to access rights administration uses privilege graphs (PG) which capture functionality, structure and authorizations. A PG (figure 2) is an acyclic graph with three types of nodes: functionality, role and user/group. A path from a given user node to a functionality node means that the user is authorized to execute the functionality. The

[2]Separation of duty [CW87], on the other hand, ensures that the supervisor does not perform both roles.

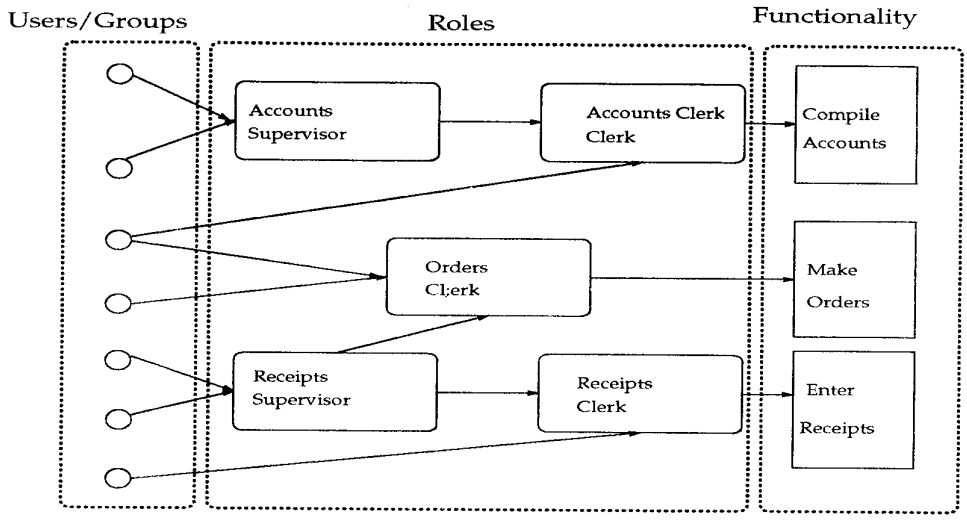


Figure 2: Baldwin's Privilege Graph

access rights available to such a user are all the privileges specified in roles on any such path. Ting et.. al.'s [TDH92] approach utilizes hierarchical ordering of roles in which for any given roles in a path, those lower in the hierarchy have lower functionality than those high in the hierarchy. In general, the path captures a subsetting relationship between the roles such that for a given directed edge $\langle v_i, v_j \rangle$, $\Psi(v_j) \subseteq \Psi(v_i)$. Both of these structures have what we term the *acyclicity property.*

Definition 5 *Acyclicity Property: A role organization structure is said to have the acyclicity property if in a graph of the role relationships, with the roles as nodes, we have a directed edge $\langle r_i, r_j \rangle$ whenever $\Psi(r_i) \subseteq \Psi(r_j)$ and the graph is acyclic.* □

Property 1 *Role Organization Structure Acyclicity: A role organization must preserve the acyclicity property in order to offer differentiated access to system information via role-based protection techniques.* □

3 Modeling Role Organization

A role is a collection of privileges which facilitates the execution of some *functionality* for an authorized user. Roles in a system can have different kinds of relationships among them based on their associated functionalities and organizational constraints. Thus it is important to develop some formal organizational framework which expresses desirable properties for an enterprise whose security is being enforced and, in the process, captures the relationships among roles. Such a framework will facilitate the analysis of privilege distribution and sharing.

In this section, we discuss and model basic role relationships which form the basis of a role organization framework. We start with relationships between two roles and introduce the concepts of the minimum and maximum privilege sets in a role-based system and their relationship with other roles. Finally, we combine these concepts to yield a framework for role organization.

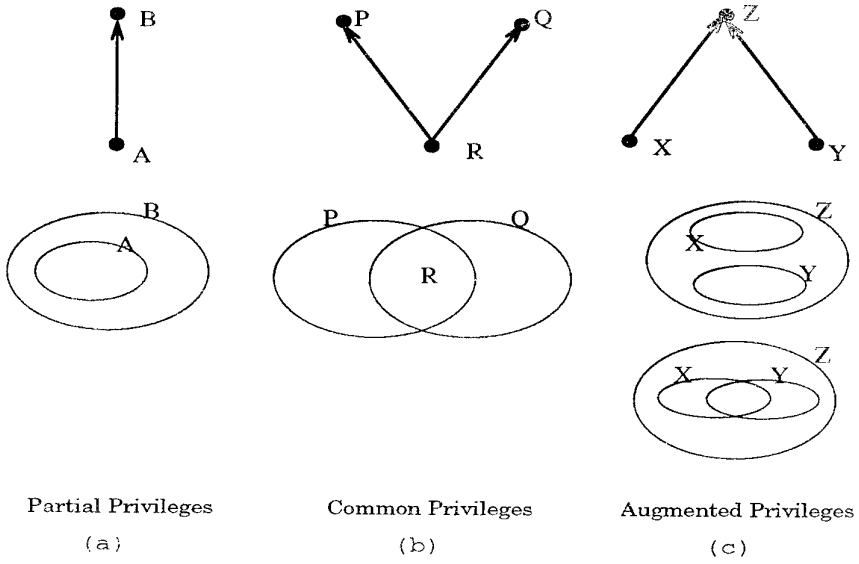


Figure 3: Three Kinds of Basic Role Relationships

3.1 Basic Role Relationships

We identify three kinds of basic relationships: *junior-senior*, *common "junior"* and *common "senior"*. The *junior-senior* relationship, expressed as **junior→senior**, captures the fact that the senior role's privileges include those of the junior one. A role is a *common junior* of two other roles if it shares some privileges with both of these senior roles. A role which encompasses all of the privileges of two junior roles is called a *common senior* to these roles. Figure 3 shows these three possibilities with Venn diagrams over the associated privileges. In all cases, there is privilege and functionality sharing between two roles.

1. **Partial Privileges**

   With partial privilege sharing, privileges defined in one role are a complete subset of privileges in another role. This implies shared functionality via the shared privileges. For instance, the **clerk** and **supervisor** roles in example 1 share the functionality associated with the **clerk** role, i.e. a user authorized to the **supervisor** role can execute the functionalities associated with both roles (figure 3a).

   We model such *direct* functionality and privilege sharing using the **is-junior** relationship denoted by "→". In our example, **clerk→supervisor**. In general, given two roles $r_i, r_j \in \mathcal{R}$ with $r_i \rightarrow r_j$, we have the following interpretation:

   > *r_i and r_j are "junior" (subservient) and "senior" (superior) roles, respectively. Moreover, r_i's privileges and functionality are available to r_j. Hence $\Psi(r_i) \subseteq \Psi(r_j)$. We say r_i's privileges are* **indirectly** *available to r_j.*

   Definition 6 *is-junior relationship ($\rightarrow$): An* is-junior *relationship exists between two roles r_i and r_j, denoted $r_i \rightarrow r_j$, if and only if $\Psi(r_i) \subseteq \Psi(r_j)$.* □

   The *is-junior* relationship can be seen as a role-role authorization in which the superior role is authorized to the privileges of the junior role.

   If we consider relative authority as a measure of the privileges associated with a role, then the *is-junior* relationship can be seen as specifying which of the two roles has a

higher authority than the other. In our case the junior role exercises less authority than the superior one. Moreover, the *is-junior* relationship can be seen as specifying the flow of authority in which the senior role exercises more authority than the junior one. Further, for this authority to be meaningful, this relationship must be *acyclic*; it must preserve property 1.

2. **Common Privileges**

Another form of relationship between two roles is where there is privilege sharing in which roles have a non-empty intersection of their privilege sets but with neither of the sets being a subset nor a superset of the other. Such a relationship can be used to express an overlap of responsibility (figure 3b).

If there exists a role defined whose privilege set is some or all of this intersection, then we say such a role is a **common-junior** of the other two roles. We denote the *common-junior* relationship by "$\odot$". In general, $r_i \odot r_j$ is not unique. Suppose we have roles A, B and C related as $C \in A \odot B$. Suppose the privilege sets associated with A and B are $\Psi(A)$={1,2,3,4} and $\Psi(B)$ ={3,4,5,6,7}, respectively. $\Psi(C)$ must be a common subset of both $\Psi(A)$ and $\Psi(B)$, i.e. $\Psi(C) \subseteq (\Psi(A) \cap \Psi(B)) = \{3,4\}$.

In general, given three roles $r_i, r_j, r_k \in \mathcal{R}$ and $r_k \in r_i \odot r_j$, we have the following interpretation:

> *both r_i and r_j are senior (superior) roles to r_k. Moreover, r_k's privileges and functionality are indirectly available to both r_i and r_j. Hence $\Psi(r_k) \subseteq \Psi(r_i)$ and $\Psi(r_k) \subseteq \Psi(r_j)$.*

Definition 7 *common-junior relationship ($\odot$): Given roles $r_i and r_j$, $r_i \odot r_j$ is all r_k such that $\Psi(r_k) \subseteq (\Psi(r_i) \cap \Psi(r_j))$.* □

3. **Privilege Augmentation**

Another important consideration is privilege augmentation. In analyzing privilege distribution it may be necessary to find a role that embodies the functionality and privileges of two given roles. Such a role's privileges will be a superset of both given roles (figure 3c).

The relationship in such a case is termed **common-senior** and is denoted by "$\oplus$". In general, $r_i \oplus r_j$ is not unique. Suppose we have roles X, Y and Z related as $Z \in X \oplus Y$. Let $\Psi(X)$={1,2,3,4} and $\Psi(Y)$={6,7,8,9}. For Z's privileges to be a common superset of those of X and Y, we must have $(\Psi(X) \cup \Psi(Y)) \subseteq \Psi(Z)$, i.e. {1,2,3,4,6,7,8,9} $\subseteq \Psi(Z)$.

Given three roles $r_i, r_j, r_k \in \mathcal{R}$ and $r_k \in r_i \oplus r_j$, we have the following interpretation:

> *both r_i and r_j are junior (subservient) roles to r_k. Moreover, both r_i's and r_j's privileges and functionalities are indirectly available to r_k. Hence $\Psi(r_i) \subseteq \Psi(r_k)$ and $\Psi(r_j) \subseteq \Psi(r_k)$.*

Definition 8 *common-senior relationship ($\oplus$): Given roles r_i and r_j, $r_i \oplus r_j$ is all r_k such that $(\Psi(r_i) \cup \Psi(r_j)) \subseteq \Psi(r_k)$.* □

The foregoing relationships can be extended to cater for more than two roles.

1. **Partial Privilege Sharing**

From the definition of the *is-junior* relationship, if $(r_i \rightarrow r_j)$ and $(r_j \rightarrow r_k)$ then it must also be true that $r_i \rightarrow r_k$ since $(\Psi(r_i) \subseteq \Psi(r_j)) \wedge (\Psi(r_j) \subseteq \Psi(r_k)) \Rightarrow (\Psi(r_i) \subseteq \Psi(r_k))$. This then captures the *transitive* property of the *is-junior* relationship. In general, if we have a role relationship of the form: $r_i \rightarrow r_{i+1} \rightarrow \cdots \rightarrow r_{i+n}, n \geq 0$, it follows that $\Psi(r_i) \subseteq \Psi(r_{i+1}) \subseteq \cdots \subseteq \Psi(r_{i+n})$. This captures the monotonic increasing property of the privilege function for roles related via the *is-junior* relationship.

Property 2 *The privilege function Ψ increases monotonically with respect to the* is-junior *$(\rightarrow)$ relationship.* □

We denote $r_i \rightarrow r_{i+1} \rightarrow \cdots \rightarrow r_{i+n} \rightarrow r_j$ by $r_i \rightarrow^* r_j$ for $n \geq 0$ and $r_i \rightarrow^+ r_j$ for $n > 0$. This leads to the concept of a path:

Definition 9 *Role Path: A role path, p, between two roles r_i and r_j is of the form $r_i \rightarrow^* r_j$. A trivial path exists between a role and itself.* □

Other properties of the *is-junior* relationship include *reflexivity* and *antisymmetry*. Given roles r_i and r_j, we have $r_i \rightarrow r_i$ (reflexivity) since $\Psi(r_i) \subseteq \Psi(r_i)$. As well, $((r_i \rightarrow r_j) \wedge (r_j \rightarrow r_i)) \Rightarrow r_i = r_j$. This follows from the observation that $(r_i \rightarrow r_j) \Rightarrow \Psi(r_i) \subseteq \Psi(r_j)$ and $(r_j \rightarrow r_i) \Rightarrow \Psi(r_j) \subseteq \Psi(r_i)$. With $\Psi(r_i) \subseteq \Psi(r_j)$ and $\Psi(r_j) \subseteq \Psi(r_i)$ and by the acyclicity property, it follows that $\Psi(r_i) = \Psi(r_j)$, which implies $r_i = r_j$. This is the basis of the following property:

Property 3 *Role Privilege Set Uniqueness: A Role's privilege set must be unique.* □

2. **Common Privileges**

From the *common-junior* $(\odot)$ relationship above, observe that the common subset of two roles need not be an immediate junior role of both roles in question. The following lemma expresses the relationship between the *is-junior* and the *common-junior* operators, $\rightarrow$ and $\odot$, respectively:

Lemma 1 *If $r_k \in r_i \odot r_j$, then $r_k \rightarrow^+ r_i$ and $r_k \rightarrow^+ r_j$.* □

The *common-junior* operator $(\odot)$ is commutative, associative and reflexive; i.e. $r_i \odot r_j = r_j \odot r_i$, $r_i \odot (r_j \odot r_k) = (r_i \odot r_j) \odot r_k$ and $r_i \odot r_i$ is defined and includes r_i.

3. **Privilege Augmentation**

As with the *common-junior* relationship, the *common-senior* relationship need not involve immediate superiors of the role under consideration. The following lemma captures the relationship between the two operators $\rightarrow$ and $\oplus$:

Lemma 2 *If $r_k \in r_i \oplus r_j$, then $r_i \rightarrow^+ r_k$ and $r_j \rightarrow^+ r_k$.* □

The operation $\oplus$ is commutative, associative and reflexive i.e. $r_i \oplus r_j = r_j \oplus r_i$, $r_i \oplus (r_j \oplus r_k) = (r_i \oplus r_j) \oplus r_k$ and $r_i \oplus r_i$ is defined and includes r_i.

3.2 The Concepts of Minimum and Maximum Privilege Sets

It is possible that an organization provides a minimum set of privileges available to every user. Such a basic privilege set, for instance, can be things like the ability/permission to log onto a computer system, the privilege to get into certain areas of an organization's premises, etc. In general, this minimum privilege set represents the very minimum that any valid user can be authorized to.

Since users are authorized to specific roles, it is possible to organize such a basic set of privileges into a role such that they are available via explicit authorization or via role relationships with other roles. We denote the role with the basic privilege set **MinRole**. In general, depending on a particular organization, **MinRole**'s privilege set can be empty.

$$\Psi(\text{MinRole}) = \begin{cases} \text{Minimum mandatory privilege set} & \text{if defined} \\ \emptyset & \text{otherwise} \end{cases}$$

For all $r \in \mathcal{R}, \text{MinRole} \rightarrow^{+} r$ holds.

Property 4 *Minimum Privilege Property:* **MinRole** *is always defined.* □

With the introduction of **MinRole**, there is always at least one *common-junior* for all roles, namely **MinRole**.

As with **MinRole**, we envisage **MaxRole**, some system "chief executive" role, which embodies the collection of all privileges in a given system. Theoretically, a user authorized to **MaxRole** can execute any functionality using the associated privileges in whatever role they are specified. Unlike $\Psi(\text{MinRole})$ which can be empty, $\Psi(\text{MaxRole})$ can never be empty if the system is intended to accomplish anything at all.

$$\Psi(\text{MaxRole}) = \bigcup_{r \in \mathcal{R}} \Psi(r)$$

For all $r \in \mathcal{R}, r \rightarrow^{+} \text{MaxRole}$ holds.

Property 5 *Maximum Privilege Property:* **MaxRole** *is always defined.* □

With the introduction of **MaxRole**, there is always at least one *common-senior* for two roles, namely **MaxRole**.

The *is-junior*, *common-junior* and *common-senior* relationships introduced in the previous section capture all manner of relationships that can be used to associate two or more roles when there is need for analysis of their interaction. **MinRole** and **MaxRole** express the concepts of minimum mandatory and maximum privilege sets, respectively, in a system. Combining these yields representations such as those in figure 4.

For the purposes of security and the need for dispersion of powers, MaxRole may not be authorized to any one individual in an organization. In an ideal situation, MaxRole conceptually corresponds with the role of a Chief Executive in an organization. It is unlikely that an administrative or a security policy would advocate such singular exercise of powers. Moreover, there is a very realistic risk that allowing exercise of privileges of MaxRole can compromise the system. However, such problems need not arise if we make the exception that no single user can exercise the privileges of MaxRole. This will make MaxRole a non-executable role. Other policies may choose a collective execution of the role, e.g. by a number of votes of authorized users. Whatever the case, authorization to MaxRole with be a matter of a specific security policy. MaxRole, in our modeling, is useful for purposes of completeness. It ensures that every two roles in the system have a common-senior just as MinRole ensures that every two roles have a common-junior.

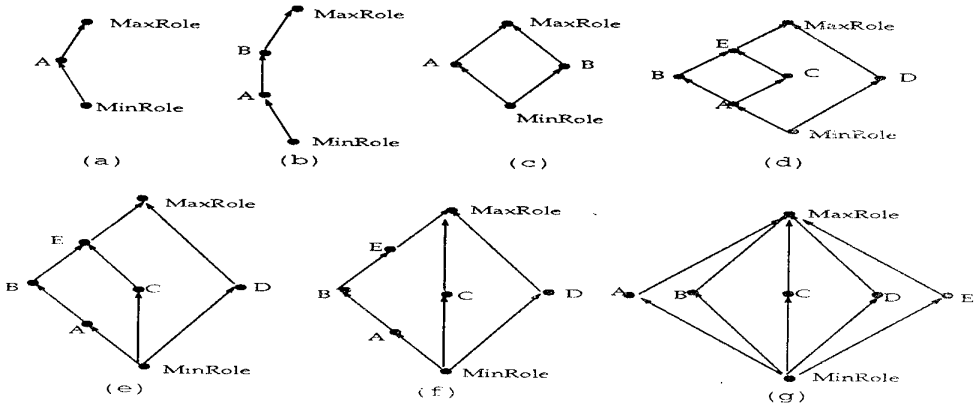


Figure 4: Different Forms of Role Organization

4 A Role Graph Model for Role Organization

The basic role relationships discussed in section 3 point to an acyclic role graph organization for roles. In this section we develop the modeling further using graph theory. We present a role graph model for role organization and develop algorithms for the management of roles and their relationships.

4.1 The Model: Informally

To minimize the task of enumerating the privileges of each role, we organize them using the concepts introduced in section 3 which incorporate acyclicity of the role graph structure and the monotonicity of role privileges for any path. Such a structure, along with rules for role ordering and determining the privileges associated with a role, facilitate a simple, yet elegant, organization of roles to reflect the *authority*[3] attached to each role. Role ordering and role inter-relationships, in turn, offer a means of distributing privileges among the roles. The idea is that we explicitly assign a privilege at the lowest point in the role graph where it is desirable. Since our formulation specifies that high order roles can execute the privileges of the lower order ones with a connecting path, we can make the least number of explicit privilege assignments that would facilitate the desired distribution.

From the ordering, we define *authority paths* that are linear (total) orders of roles according to increasing authority, connected by the *is-junior* ($\rightarrow$) relationship which can be seen to be specifying the flow of authority. In essence, the ordering asserts the fact that higher authority roles have access to more privileges than lower ordered ones in any given path. The effective privileges associated with a role result from those privileges *directly* associated with the role and those *indirectly* associated with it. The former are those privileges *explicitly* specified in the role while the latter are those privileges specified in lower order roles connected by a path to the role.

4.2 The Role Graph Model: Formally

This section presents the formal organization of roles into a role graph $RG = (\mathcal{R}, \rightarrow)$, as shown in figure 5. The nodes of the graph correspond to the roles given, and include **MaxRole** and

[3]Our use of this term will become clear as we advance.

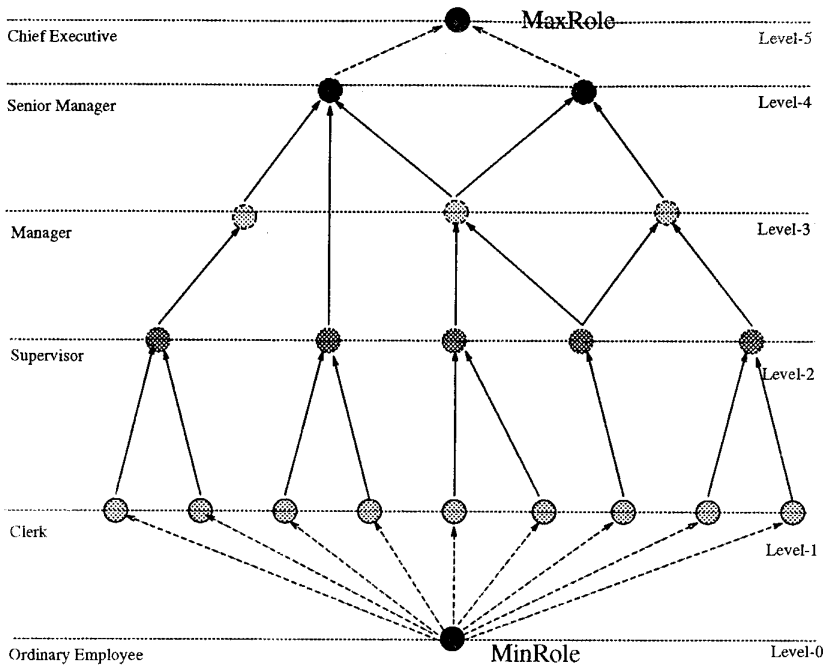


Figure 5: Example of Role Graph

MinRole. $\mathcal{R} = \{r_1, r_2, \cdots, r_n, \text{MaxRole}, \text{MinRole}\}$. The edges are defined by the *is-junior* relationship. Note that by the definition of privileges for **MaxRole** and **MinRole** and the definition of *is-junior*, there is an edge from **MinRole** to every r_i, and an edge from every r_i to **MaxRole**. The common-junior and common-senior relationships, ($\odot$ and $\oplus$) still have the same meaning as previously.

Note that if a system administrator is specifying roles, it is possible that the privileges are specified in a highly redundant fashion. In other words, rather than specifying the minimum set of direct privileges for a role, some indirect privileges might be given as being direct privileges. The function $\Psi(r)$ returns the set of all direct and indirect privileges of a role, which we also call the *effective* privileges. The version of the graph which we will present to the role administrator should *neither* have redundant privilege specifications *nor* redundant *is-junior* relationships (i.e. redundant graph edges), in order to highlight the true nature of the role relationships. We will further explain this reduced form of the graph shortly.

Paths in the role graph not involving **MaxRole** and **MinRole** are of more interest to us. Consequently, we shall use the following role graph path definition in the subsequent sections.

Definition 10 *Role Graph Path: A role graph path, p, is of the form $r_i \rightarrow r_{i+1} \rightarrow \cdots \rightarrow r_{i+n} \rightarrow r_j, n \geq 0$ such that $r_i \neq MinRole \wedge r_j \neq MaxRole$.* □

The quadruple $(\mathcal{R}, \rightarrow, \oplus, \odot)$ which includes **MaxRole** and **MinRole**, specifies an *authority structure* for roles. For any role graph path of the form $r_i \rightarrow \cdots \rightarrow r_n, n \geq 1$ we have an authority relation of the form $r_1 < \cdots < r_n$, with the authority embodied in the roles on a path totally ordered. In general, given any two roles $r_1, r_2 \in \mathcal{R}$, $r_1 < r_2$, $r_2 < r_1$ or they are incomparable. Where there is a path (call it an *authority flow path*), the roles in the path form a total order.

Definition 11 *Path Role Set: The role set of a given path, denoted by $\Gamma(p)$, is the set of all roles that compose the path. We say that a given role participates in a path if it belongs to the path's role set.* □

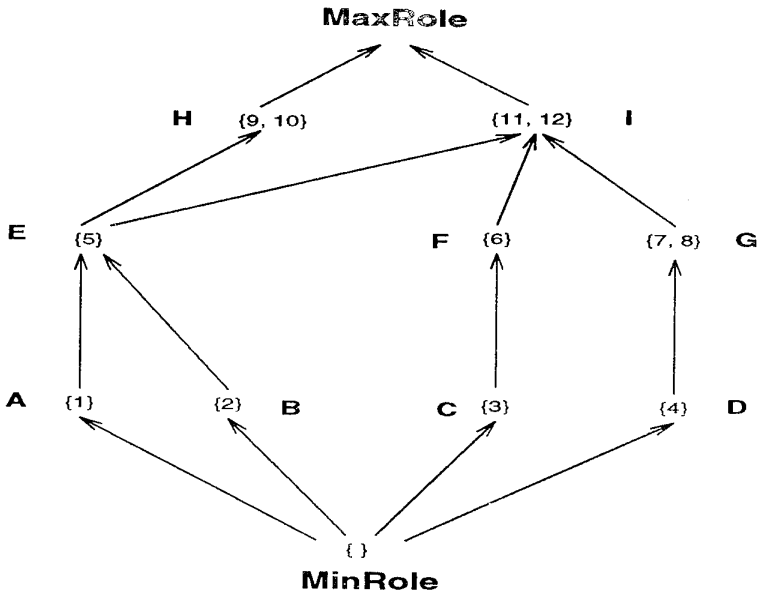


Figure 6: Role Graph with Privileges

Privilege Distribution Table For Figure 6			
Role Name	Direct (D)	Indirect (I)	Effective ($D \cup I$)
A	{1}	{ }	{1}
B	{2}	{ }	{2}
C	{3}	{ }	{3}
D	{4}	{ }	{1}
E	{5}	{1,2}	{1,2,5}
F	{6}	{3}	{3,6}
G	{7,8}	{4}	{4,7,8}
H	{9,10}	{1,2,5}	{1,2,5,9,10}
I	{11,12}	{1,2,3,4,5,6,7,8}	{1,2,3,4,5,6,7,8,11,12}

Table 1: Table of Privileges

We extend the function Ψ to paths as follows: for a path p, $\Psi(\Gamma(p)) = \bigcup_{r_i \in p} \Psi(r_i)$.

Definition 12 *Path Independence: Let p_i and p_j be two paths in a role graph. We say p_i is independent of p_j if $\Psi(\Gamma(p_i)) \cap \Psi(\Gamma(p_j)) = \Psi(MinRole)$.* □

In other words, the two role sets are related only via **MaxRole** and **MinRole**. Such independence can be exploited to prohibit privilege sharing by ensuring that the privilege sets of two independent paths are disjoint.

Example 2 Consider figure 6 where we have distinct privileges numbered $1, \cdots, 12$ with privileges $1, \cdots, 6$ directly assigned to roles $A, \cdots, F$ and $\{7,8\}, \{9,10\}, \{11,12\}$ assigned roles G, H, I respectively. We have a role graph specification as follows: $A \rightarrow E, B \rightarrow E, C \rightarrow F, D \rightarrow G, E \rightarrow \{H, I\}, \{F, G\} \rightarrow I$, MaxRole $= H \oplus I$, and MinRole $= A \odot B \odot C \odot D$ with $\Psi(\text{MinRole}) = \emptyset$.

From this we can compute the privileges of various roles and obtain the privileges distribution as in table 1. Moreover, we have the following relationships relating to the $\odot, \oplus, \rightarrow$ operators:

1. The *common-junior* operator, $\odot$, defines a common subset of privileges for any two roles. Consider $E \in H \odot I$ and note that $\Psi(H \odot I) = \Psi(E) = \{1,2,5\} = \Psi(H) \cap \Psi(I)$.

2. The *common-senior* operator, $\oplus$, defines the union of privileges of two roles and as such is a common superset for any two roles. Consider $I \in F \oplus G$ and note that $\Psi(F \oplus G) = \Psi(F) \cup \Psi(G) = \{3,4,6,7,8\} \subseteq \Psi(I) = \{3,4,6,7,8,11,12\}$.

3. The *is-junior* operator, $\rightarrow$, defines a proper subset relationship between two roles, e.g. $E \rightarrow H$. Note that $\Psi(E) = \{1,2,5\} \subset \{1,2,5,9,10\}$. This is true for all roles related via the *is-junior* relationship.

4. Paths $A \rightarrow E \rightarrow H$ and $C \rightarrow F$ are independent paths since their roles sets {A,E,H} and {C,F} are mutually exclusive and the two paths are related via only via **MaxRole** and **MinRole**.

□

The role graph in figure 6 shows only *direct* (non-redundant) privileges for each node, and has no redundant edges. Specifying a role's direct privileges and its *is-junior* relationships with other roles completely specify its effective privileges.

Definition 13 *Direct Privileges: Let $Direct(r)$ denote the direct privileges of a role; i.e. $Direct(r) \subseteq \Psi(r)$ such that for all $r_i \rightarrow r$, $\Psi(r_i) \cap Direct(r) = \emptyset$.* □

For the purpose of the algorithms below, assume that for each role in a role graph, we keep $Direct(r)$ and *is-junior* relationships. By the definition of *is-junior*, the edge set in the role graph will in fact be highly redundant. What we want to present to the role administrator, and maintain, is the transitive reduction of the graph [AGU72]. The transitive reduction of an acyclic graph is a graph in which there are no edges $r_i \rightarrow r_j$ whenever there is a path $r_i \rightarrow^+ r_j$ in the graph. Inputs to and outputs of the algorithms assume *well-formed graphs.*

Definition 14 *Role Graph Well-Formedness: A role graph is well-formed if it is a transitive reduction and if the direct privilege set associated with each role r conforms to the definition of $Direct(r)$.* □

By the original definition of the edge set (based in turn on the *is-junior* relationship which depends on the effective privilege sets of nodes), a path $r_i \rightarrow^+ r_j$ exists in the well-formed role graph whenever $\Psi(r_i) \subseteq \Psi(r_j)$. The following terms will be useful in the algorithms to be presented below:

Definition 15 *Juniors(r) The set of Junior roles for a given role r is all r_i such that $r_i \rightarrow^+ r$.* □

Definition 16 *Seniors(r)The set of Senior roles for a given role r is all r_j such that $r \rightarrow^+ r_j$.* □

Constraint 1 *Role Graph Privilege Set Invariant Constraint: The effective privilege set of every role in a role graph remains invariant unless altered by the system security officer, SSO.* □

The SSO exercises privileges like any other system user by executing in an authorized security administration role. This can be seen as the security information administration role. However, care must be taken to ensure there is no conflict of interest. Hence no one user, whether SSO or not, should be able to administer security information pertaining to one's access rights.

4.3 Role Graph Maintenance Algorithms

We are now ready to introduce some algorithms to assist a role administrator in specifying and modifying a collection of roles. These will ultimately be incorporated in a role maintenance tool.

Our goal is to have all the operations map a well-formed role graph to another well-formed role graph. We assume that the administrator begins with a graph containing only **MaxRole** and **MinRole**. Any direct privileges defined for **MinRole** can be specified at this time.

The role graph can be expanded at any point by adding new roles as need may arise while retaining the role graph structure. This strategy offers a flexible manner of introducing new privileges into the role graph. Such privileges can be incorporated into an existing role graph by introduction of new roles or by increasing the privileges of existing ones. New roles can be introduced by the addition of completely new roles, or by partitioning existing roles either horizontally or vertically. We also consider role deletion. In all these cases, we can have an increment or decrement in the overall privileges associated with paths in which the affected role participates. Such privileges can remain invariant, be reduced or be increased depending on the operation. Given the space constraints here, we address the cases where (1) path privileges are introduced with the addition of a new role, (2) path privileges may or may not remain invariant with the deletion of a role, (3) path privileges are partitioned with the horizontal partition of a role and (4) privileges remain invariant with the vertical partition of a role.

Consequently, after carrying out the operations on the graph, our procedures will confine themselves with the immediate neighbourhood of the target role. In other words we look for redundant arcs generated due to the operation in question. This involves the immediate senior and immediate junior role sets of the roles affected by the operations.

4.3.1 Role Addition & Deletion

By role addition we mean the creation and incorporation of a totally new role into the role graph. Such a role is defined (name and privilege set) before being integrated into the role graph. While the integration process must preserve the role definition, it is important to ensure that if there are privileges defined in the new role that exist in junior roles in the target paths, they must be removed to take away the redundancy. To introduce such a role requires the specification of the target paths and the position in the paths. This involves the specification of the target superior and junior role(s) for the role to be added (see figure 7a).

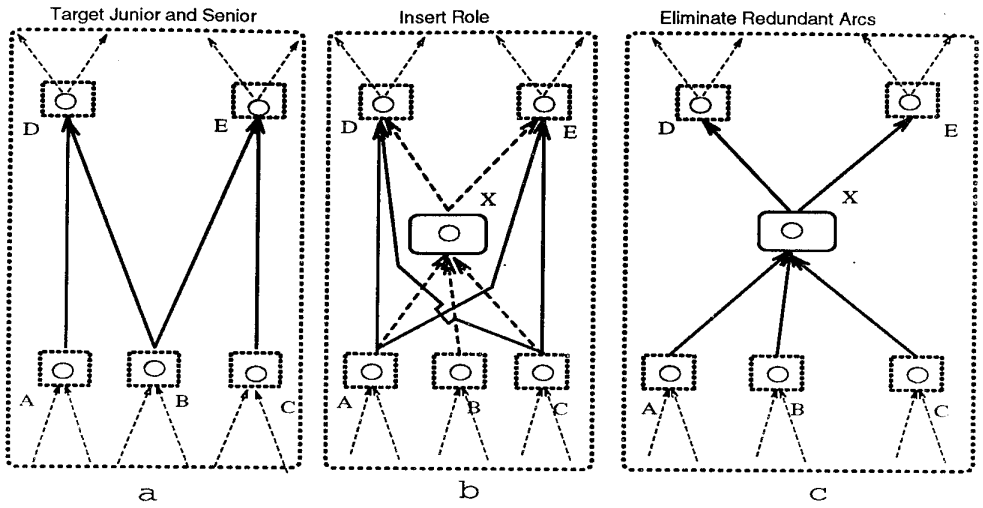


Figure 7: Role Addition

The role to be inserted is added to the node set of the graph, and the appropriate edges are created to indicate the immediate junior and superior roles. It is possible that a node already exists with the same effective privilege set. Once this possibility has been eliminated, redundant paths are removed from the resulting structure. Finally, privilege resolution is done to remove redundant privileges from *Direct* of the new node, and privileges in nodes in Seniors of the the new node made redundant by this insertion. Note that for a node r, the set Seniors(r) can be enumerated by a depth-first search in the role graph starting at node r [CLR90, Man89]. Similarly, the set Juniors(r) can be computed by a depth-first search of the graph formed by reversing the edges in the role graph, again starting the search at node r. The details of these operations will not be given here.

Algorithm 1 in figure 8 and also figure 7 illustrate the role addition process.

The flip side of *role addition* is *role deletion* which involves the elimination of a role from the role graph. This process requires specifying the target role and short-circuiting it by making the target's immediate subservient role(s) the immediate subservient role(s) of the target's immediate superior(s). In doing so, the privileges associated with the deleted role can either be eliminated or distributed. Privilege elimination involves overall privilege reduction of the path associated with the role so deleted.

Retaining the privileges of the deleted role, on the other hand, requires a specification of how these privileges will be distributed among the existing roles. It is reasonable to assume that such role deletion would not affect the effective privilege sets of any superior roles of the deleted role. Hence such privileges must be transferred to the immediate superiors. This would ensure path privilege invariance. This case is illustrated pictorially in figure 9. See the associated algorithm 2 of figure 10.

Example 3 Suppose our target role for deletion is role D in figure 9a with the constraint that all existing paths must keep their privilege sets invariant. For this purpose we choose to shift the privilege set of the target role to its superiors.

To achieve this, first transfer the privileges from role D to both F and G which are both superior to D. This results in roles roles FX and GX which we make immediate superiors of both A and B. The previous edges incident to role D, i.e. $A \rightarrow D \rightarrow F, A \rightarrow D \rightarrow G, B \rightarrow$

Algorithm 1 Role_Addition(rg, target, s.target.set,j.target.set)

```
/* For the addition of a given role into a role graph */
Input: rg = ⟨R, →⟩ (the role graph), target role to be added (role name along with its proposed direct privilege set),
   s.target.set (immediate superior set for the target), j.target.set (immediate junior set for the target),
Output: The role graph with target added and overall privileges of other roles left intact.
   Var r, r_j, r_s: roles;
   Begin
        If ∃(r_s →+ r_j) for any r_s ∈ s.target.set, r_j ∈ j.target.set
            Then abort                                          /* Must not violate acyclicity */
        Else Begin
            1.  Ψ(target):= (∪_{r∈j.target.set} Ψ(r)) ∪ Direct(target);
                    /* Compute the effective privileges of target role */
            2   If Ψ(r) = Ψ(target) for any r ∈ R
                    Then target := r.                           /* Role privilege sets must be unique */
            3.  If ∃(target →+ r_j) for any r_j ∈ j.target.set
                    Then abort                                  /* Must not violate acyclicity */
                Else Begin
                    a.  R := R ∪ target.                        /* Add target to system roles */
                    b.  For all r_s ∈ s.target.set do add the edge target → r_s;
                    c.  For all r_j ∈ j.target.set do add the edge r_j → target;
                    d.  If for any r ∈ R, Ψ(r) ⊂ Ψ(target) and NOT(r →+ target)
                            Then add the edge r → target.       /* Add this inferred edge */
                    e.  If for any r ∈ R, Ψ(target) ⊂ Ψ(r) and NOT(target →+ r)
                            Then add the edge target → r.       /* Add this inferred edge */
                    f.  Rem_Red_Arcs(rg, j.target.set, s.target.set, target);
                    g.  Red_Priv_Res(rg, j.target.set, target);
                end;
            4.  For all r, r_i, r_j ∈ R if Ψ(r_i) = Ψ(r_j) then     /* Remove any duplicate roles */
                Begin for all r_i → r do add the edge r_j → r
                    for all r → r_i do add the edge r → r_j
                    Delete all edges r_i → r and r → r_i;
                    Remove r_i; end;
        end.
   end.                                                         /* Role_Addition */

   Procedure Rem_Red_Arcs(var rg: role graph; j.target.set, s.target.set: role_set; target: role );
   /* Removes redundant arcs in the immediate neighbourhood of target role */
   Var r_k, r_j, r_s: roles;
   Begin   1.  For all r_j ∈ j.target.set do                     /* Remove direct paths where there is another path */
                   if ∃(r_j → r_k → ··· → target) then
                       Delete the edge r_j → target             /* delete the direct edge*/
           2.  For all r_s ∈ s.target.set do
                   if ∃(target → r_k → ··· → r_s) then
                       Delete the edge target → r_s             /* delete the direct edge*/
   end.                                                         /* Red_Red_Arcs */

   Procedure Red_Priv_Res(var rg: role graph; j.target.set, s.target.set: role_set; target: role);
   Var pv: privilege, r: role;
   Begin   1. For all r in Seniors(r) do                        /* remove redundant privileges from senior roles. */
                  For all pv ∈ Direct(target) do
                      if pv ∈ Direct(r) then
                          Direct(r) := Direct(r) − pv.
           2. For all r in j.target.set do                      /* remove redundant privileges from Direct(target) */
                  For all pv ∈ Ψ(r) do
                      if pv ∈ Direct(target) then
                          Direct(target) := Direct(target) − pv;
   end.                                                         /* Red_Priv_Res */
```

Figure 8: Algorithm for Role Addition

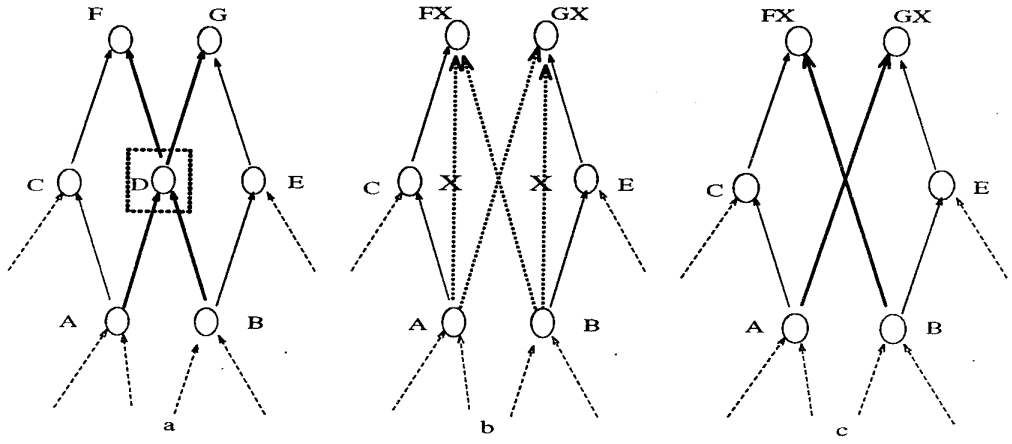


Figure 9: Role Deletion

$D \rightarrow F$, and $B \rightarrow D \rightarrow G$ are replaced by $A \rightarrow FX, A \rightarrow GX, B \rightarrow FX$, and $B \rightarrow GX$, respectively.

The next move is to do away with redundant alternative paths (marked X in the figure 9b) and remove them. We notice that paths $A \rightarrow C \rightarrow FX$ and $B \rightarrow E \rightarrow GX$ contain the set of privileges of paths $A \rightarrow FX$ and $B \rightarrow GX$, respectively. This results in a new role graph structure as shown in figure 9c. □

Both role addition and deletion correspond to real life situations where in creating a new portfolio, a new role is added while in eliminating some "office", a role will be deleted. Role deletion without privilege reduction entails elimination of some "office" in an organization while retaining the total functionality. Privileges of the deleted role would be distributed to other roles.

4.3.2 Role Partition

A role can be partitioned into two or more roles in our role graph. Essentially, the basic partition operations are either vertical or horizontal, and can of course be combined. In both cases it must be specified what the new roles and their corresponding privileges are. Where the order of "seniority" is required, as in the case of vertical partition, it must be specified as well.

In **vertical role partition**, a role is split into two or more roles and an ordering is imposed on them with the *is-junior* relationship. In doing vertical partition, we must specify the target role, the new roles to be created, their direct privileges and their ordering (according to partial privilege criterion). For instance, a role X is not only partitioned into roles $X_1, \cdots, X_n$ but also, these roles must be ordered, e.g. $X_1 \rightarrow \cdots \rightarrow X_n$ (see figure 11b and algorithm 3 of figure 12). Privilege distribution among the new roles is constrained by the privileges associated with the role being partitioned; there *must not* be an increment or decrement of privileges, i.e.

$$Direct(X) = \bigcup_{i=1,\cdots,n} Direct(X_i)$$

Consequently, the privileges associated with the paths in which the role appears neither

Algorithm 2 Role_Deletion(rg, target, inv)

```
/* Deletes a specified role retaining or discarding its privileges depending on inv */
Input: rg = (R, →) (the role graph structure), target (the target role to be deleted),
       inv Boolean indicating whether or not to retain the role's privileges
Output: The role graph structure with target deleted

Var s_set, j_set: role set; r, r_j, r_s: role;
    Begin 1. s_set := Superior_Set(target);                    /* Get the senior set */
          2. j_set := Junior_Set(target);                      /* Get the junior set */
          3. For all r_s ∈ s_set do                            /* Connect Junior and Senior Roles */
                For all r_j ∈ j_set do add r_j → r_s;
          4. If inv then do
                For all r_s ∈ s_set do                         /* Transfer Privileges to superiors */
                   Direct(r_s) := Direct(r_s) ∪ Direct(target);
          5. For all r_s ∈ s_set do                            /* Remove all redundant arcs */
                For all r_j ∈ j_set do
                   If ∃(r_j → r_k ··· → r_s) then delete r_j → r_s;
          6. R := R − target;                                  /* Take out target from system roles
    end                                                        /* Role_Deletion */

    Function Superior_Set(var rg: role graph; target: role): role_set;
    Var Tempset: role_set; r: role;
    Begin 1. Tempset := ∅;
          2. For all r with target → r do
                Tempset := Tempset ∪ r;
          3. Superior_Set := Tempset
    end.                                                       /* Superior_Set */

    Function Junior_Set(var rg: role graph; target: role): role_set;
    Var Tempset: role_set, r: role;
    Begin 1. Tempset := ∅;
          2. For all r with r → target do
                Tempset := Tempset ∪ r;
          3. Junior_Set := Tempset
    end                                                        /* Junior_Set */
```

Figure 10: Algorithm for Role Deletion

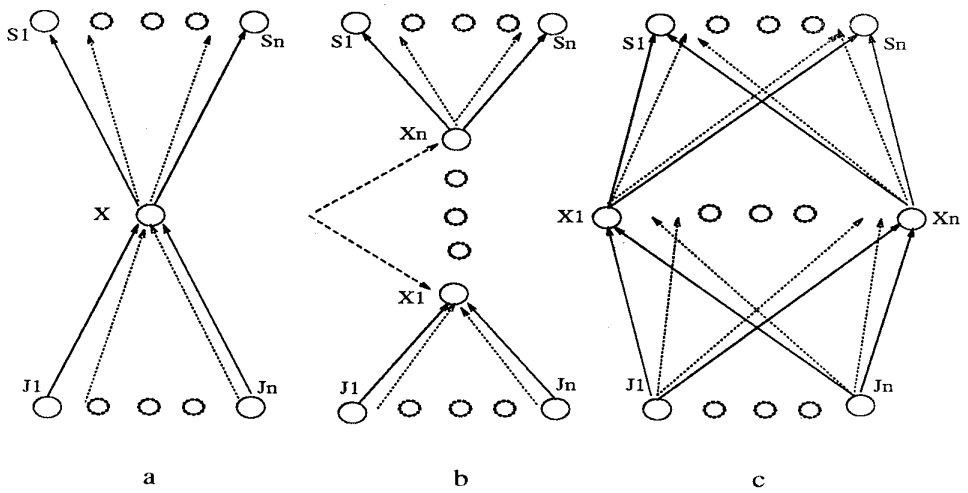


Figure 11: Vertical & Horizontal Role Partition

decrease nor increase. In general, vertical partition leaves the privilege set associated all paths unaffected; only the path length increases.

Further constraints include the requirement for distinct direct privilege sets for the newly created roles, i.e. for any

$$X_i, X_j \in \{X_1, \cdots, X_n\}, Direct(X_i) \bigcap Direct(X_j) = \emptyset$$

Suppose we have a target role for partition (call it X) with a relationship $\{J_1, \cdots, J_n\} \rightarrow X \rightarrow \{S_1, \cdots, S_n\}$ which is partitioned vertically into roles $\{X_1, \cdots, X_n\}$ such that $\{J_1, \cdots, J_n\} \rightarrow \{X_1 \rightarrow \cdots \rightarrow X_n\} \rightarrow \{S_1, \cdots, S_n\}$. It follows that $(X_n \subseteq (S_1 \odot S_2 \odot \cdots \odot S_n)) \wedge (X_1 \subseteq (J_1 \oplus J_2 \oplus \cdots \oplus J_n))$.

Horizontal role partition, on the other hand, involves partitioning a role into two or more roles with none of them being subservient (superior) to another (see figure 11c and algorithm 4 of figure 13). Partition, as used here, merely distributes the direct privileges of the target role among newly created roles that replace it. In partitioning a role, there should be no effective increment or decrement of privileges. In other words, as with vertical partitioning, if role X is partitioned into roles $X_1, \cdots, X_n$, we require that

$$Direct(X) = \bigcup_{i=1,\cdots,n} Direct(X_i)$$

The direct privilege sets of these newly created roles can have empty or non-empty intersections. However, none of them should have identical privilege sets. Note that, unlike vertical partition, horizontal partition can cause a variation of privileges associated with a path when the target role is the senior-most role in the path.

Suppose we have a target role for partition (call it X) with a relationship $\{J_1, \cdots, J_n\} \rightarrow X \rightarrow \{S_1, \cdots, S_n\}$ which is partitioned horizontally into roles $\{X_1, \cdots, X_n\}$ such that $\{J_1, \cdots, J_n\} \rightarrow \{X_1, \cdots, X_n\} \rightarrow \{S_1, \cdots, S_n\}$. It follows that $(\{J_1, \cdots, J_n\} \subseteq (X_1 \odot X_2 \odot \cdots \odot X_n)) \wedge (\{S_1, \cdots, S_n\} \subseteq (X_1 \oplus X_2 \oplus \cdots \oplus X_n))$

Updates to the role graph include the reduction and addition of role privileges which require the specification of the target role and privileges to be removed/added, but do not alter the basic structure and relationships in the role graph structure. These may be addressed within the context of role-privilege authorization.

Algorithm 3 Vertical_Partition(rg, target, $\{((x_i, \rightarrow), x_i.rpset_i)\}$)

```
/* Partitions a given role vertically */
Input: rg = (R, →) (the role organization structure); target (the target role to be partitioned),
       {((x_i, →), rpset_i)} (the new role-direct privilege set pairs and their ordering).
Output: The role graph with target vertically partitioned into {((x_i, →), rpset_i)} and integrated
        into the role graph structure.
Uses Superior_Set and Junior_Set of algorithm 2 in figure 10.

Var s.set, j.set: role set; r_j, r_s: roles;

    Begin
        If Direct(target) ≠ ⋃ (Direct(x_i))
            Then abort /* Must keep privilege set invariant */
        Else Begin
            1. R := R ∪ {x_i};                          /* Add new roles to system */
            2. s.set := Superior_Set(target);           /* Generate superior set */
            3. j.set := Junior_Set(target);             /* Generate Junior set */
            4. Add edges x_1 → x_2, x_2 → x_3, ..., x_{n-1} → x_n; /* Create a Path as specified */
            5. For all x_i do Direct(x_i) := rpset_i;   /* Assign the appropriate privileges */
            6. For all r_s ∈ s.set do add x_n → r_s;    /* Join the Senior end */
            7. For all r_j ∈ j.set do add r_j → x_1;    /* Join the Junior end */
            8. R := R − target;                         /* Delete target from system */
        end;
    end                                                 /* Vertical_Partition */
```

Figure 12: Algorithm for Vertical Partition

Algorithm 4 Horizontal_Partition(rg, target, $\{(x_i, x_i.rpset_i)\}$)

```
/* Partitions a given role horizontally */
Input: rg = (R, →) (the role organization structure), target (the target role to be partitioned), {(x_i, rpset_i)}
       (the new role-direct privilege pairs to replace target)
Output: The role structure with target horizontally partitioned into {(x_i)} and integrated into rg.
Uses Superior_Set and Junior_Set of algorithm 2 in figure 10.

Var s.set, j.set: role set, r, r_j, r_s: roles;
    Begin If Direct(target) ≠ ⋃ (Direct(x_i))
            Then abort                                  /* Must keep privilege set invariant */
        Else begin
            1. R := R ∪ {x_i};                          /* Add new roles to system Roles */
            2. s.set := Superior_Set(target);           /* Generate the superior set */
            3. j.set := Junior_Set(target);             /* Generate the junior set */
            4. For all x_i ∈ {x_1, ..., x_n} do         /* assign the privilege set to the new role */
                   Direct(x_i) := rpset_i;              /* Assign respective privilege sets */
            5. R := R − target;                         /* Delete target */
            6. For all x_i ∈ {x_1, ..., x_n} do
               begin For all r_s ∈ s.set do add x_i → r_s;   /* Link New Roles to seniors */
                     For all r_j ∈ j.set do add r_j → x_i;   /* Link New Roles to juniors */
               end;
            7. For all r, r_i, r_j ∈ R if Ψ(r_i) = Ψ(r_j) then   /* Remove any duplicate roles */
               Begin
                     for all r_i → r do add the edge r_j → r
                     for all r → r_i do add the edge r → r_j
                     Delete all edges r_i → r and r → r_i;
                     Remove r_i;
               end
        end;
    end                                                 /* Horizontal_Partition */
```

Figure 13: Algorithm for Horizontal Partition

4.4 The Role Graph & Role Coupling

Considering our role graph model proposed in section 4.2, we term the extent of *linkage* between roles a *coupling* which is related to the extent to which privileges are shared among roles. We can have a variety of cases, e.g. where each role is independent of all others or where some roles are *coupled* and hence dependent on each other.

Definition 17 *Coupling: Coupling exists between two roles r_i and r_j if $\exists r_k$ such that $r_k \in r_i \odot r_j$ and $r_k \neq MinRole$. We call r_k a coupling role between r_i and r_j.* □

Definition 18 *Role Independence: Two roles r_i and r_j are independent if and only if $r_i \odot r_j = \{MinRole\}$, i.e. their only coupling is the role common to all roles in the role graph. In other words their only greatest lower bound is MinRole.* □

Independent roles have no coupling between them.

5 Comparison with Hierarchies, Privilege Graphs & Others

The role graph model presented here can simulate a hierarchical organization. We can convert a role graph into a tree (hierarchy) and vice versa. To obtain a tree from a given role graph, we designate **MaxRole** as the root of the hierarchy and do a recursive *bread-first* or *depth-first* traversal for every node with a relationship with **MaxRole**. A given path terminates when **MinRole** is encountered which forms the leaves of all paths in the resulting tree (hierarchy). This tree contains *all* paths present in the associated role graph. In going from a tree to a role graph, we designate the root of the tree to be **MaxRole**, do a depth-first traversal of the tree and equating nodes whenever equal privileges are encountered. The resulting role graph can then be augmented with **MinRole** if necessary. The advantage with the role graph is its *compactness*, i.e. shared nodes lower in the hierarchy, need not be duplicated. This is a major advantage in that it reduces the extent to which shared privileges are scattered among roles which makes the task of tracking their use easier.

To simulate privilege graphs [Bal90], attach to every role an associated functionality that specifies the associated duty requirements/title/etc. With the role's access control list (racl) acting as the user/group node (figure 2), it is possible to determine the authorized users for any role. An authorized user's access rights are determined by the effective privilege set $\Psi(r)$ of the associated role r to which the user is authorized. Further, remove **MaxRole** and assign its explicit privileges to roles with direct partial privilege relationship with it. As well, remove **MinRole** and assign its privileges to those roles with a direct partial privilege relationship with it. The result is a privilege graph.

Finally, although this model is based on subsets with an acyclic graph, it is different from the Bell and LaPadula Model (BLPM). Moreover, although both are meant for security application, they have different approaches to realizing protection. The BLPM relies on subsets, acyclicity and is static. However, it is based on the classification of information as opposed to the execution of operations as is the case in our model. The BLPM specifies two simple operations of either read or write access depending on object classification and subject clearance. This approach realizes multilevel security. In our model, privileges represent pre-defined executions designed in a manner intended to realize certain desired functionality in a system. These operations are designed from considerations of desired system functionality. Once defined, the operations are distributed among roles in the system in the manner that suits organizational requirements. The executions can be simple reads and writes. They can

be a combination of simple reads and writes. But they can also be complex executions such methods in object-oriented programming. These operations need not merely alter or return the information relating to a given object but can also create other objects and invoke other operations.

In the BLPM, once classification has been done, access to information is governed by the simple security property and the *-property. Its specification is static. In our model, execution of privileges can cause the assignment or revocation of privileges pertaining to some role. In that respect, our model is dynamic.

6 Summary & Conclusions

It is important to have a means for role organization that reduces the complexity of privilege management in a role-based security system. This paper has presented a model for role organization derived from three basic role relationships, viz: partial, shared and augmented privileges. These lead to a role graph formulation and use of role graph theory. The model allows for the assignment of privileges in a particular role and through role relationships, we determine the extent of privilege sharing. Given the *acyclicity property*, the role graph model facilitates role partial ordering and privilege subsetting among roles. With an appropriate assignment of privileges to roles and specification of role relationships, the role graph can ease the task of access rights rights administration in a system. Our model has the expressive power of both hierarchies [TDH92] and privilege graphs [Bal90].

The issue of role administration was addressed and algorithms for role management presented. These include algorithms for role addition, deletion and split (partition). Central to role management is the concept of the change (or lack of change) of path privileges, because path privilege changes have implications for roles with indirect access to these privileges.

The concept of paths in the role graph is important in that specific types of processing can be associated with specific paths. Since there is privilege sharing among roles within a path, one can impose constraints about the order of role participation in the processing as well as separation of duty requirements. Role and path independence are important for cases with conflict of interest. Two types of processing that conflict can be associated with independent paths and by ensuring that no user is authorized for roles from both paths, we can impose conflict of interest restriction to processing.

Currently, we are involved in the implementation of a role management tool which we hope will give further insight into the applications of the role graph model in access rights administration.

Acknowledgements

This work was supported in part by a grant from the Natural Science & Engineering Research Council, NSERC, of Canada. We also thank Jim Mullin for his useful comments on an earlier draft of this paper. The anonymous referees raised a number of issues that have been useful in making clear some of our ideas. Sheila Lindsay, who worked on an implementation of the role graph, pointed out some errors in the algorithms proposed earlier. We are grateful for her comments.

References

[AGU72] A. V. Aho, M. R. Garey, and J. D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal of Computing*, 1(2):131–137, June 1972.

[Bal90] R. W. Baldwin. Naming & Grouping Privileges to Simplify Security Management in Large Databases. In *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, pages 116–132. IEEE Computer Society Press, May 1990.

[BL75] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Unified Exposition & Multics Interpretation. Technical Report MTIS AD-A023588, MITRE Corporation, July 1975.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Presss, 1990.

[CW87] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Security Policies. In *Proc. 1987 IEEE Symposium on Research in Security and Privacy*, pages 184–194. IEEE Computer Society Press, April 1987.

[Den76] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[DM89] J. E. Dobson and J. A. McDermid. Security Models and Enterprise Models. In C. E. Landwehr, editor, *Database Security II: Status & Prospects*, pages 1–39. North-Holland, 1989.

[GMP92] J. Glasgow, G. MacEwen, and P. Panangaden. A Logic for Reasoning About Security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.

[KM92] E. V. Krishnamurthy and A. McGuffin. On the Design & Administration of Secure Database Transactions. *ACM SIGSAC Review*, pages 63–70, Spring/Summer 1992.

[Law93] L. G. Lawrence. The Role of Roles. *Computers & Security*, 12(1):15–21, Feb 1993.

[Man89] Udi Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, 1989.

[NO93a] M. Nyanchama and S. L. Osborn. Role-Based Security, Object Oriented Databases & Separation of Duty. *ACM SIGMOD RECORD*, 22(4):45–51, Dec 1993.

[NO93b] M. Nyanchama and S. L. Osborn. Role-Based Security: Pros, Cons & Some Research Directions. *ACM SIGSAC Review*, 2(2):11–17, June 1993.

[NO94] M. Nyanchama and S. L. Osborn. Information Flow Analysis in Role-Based Security Systems. In *Proc. 1994 International Conference on Computing & Information (ICCI)*, May 1994.

[RBWK91] F. Rabitti, E. Bertino, D. Woelk, and W. Kim. A Model of Authorization for Next Generation Databases Systems. *ACM TODS*, 16(1):88–131, March 1991.

[RWK88] F. Rabitti, D. Woelk, and W. Kim. A Model of Authorization for Object Oriented and Semantic Databases. In *Proc. of Int'l Conference on Extending Database Technology*, March 1988.

[TDH92] T. C. Ting, S. A. Demurjian, and M. Y. Hu. Requirements Capabilities and Functionalities of User-Role Based Security for an Object-Oriented Design Model. In C. E. Landwehr and S. Jajodia, editors, *Database Security V: Status & Prospects*, pages 275–296. North-Holland, 1992.

[Tho91] D. J. Thomsen. Role-Based Application Design and Enforcement. In S. Jajodia and C. E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 151–168. North-Holland, 1991.

User Group Structures in Object-Oriented Database Authorization

Eduardo B. Fernandez, Jie Wu and Minjie H. Fernandez

Department of Computer Science and Engineering

Florida Atlantic University

Boca Raton, FL 33431

Abstract

When there are a large number and variety of users in a system, a large number of authorization rules is required to define their access rights. Because of their number and variety these rules become too difficult and cumbersome to maintain, the authorization evaluation algorithms are not efficient and their storage takes up a large amount of memory. Also, it is hard for security administrators to understand why a specific user is given a set of rights. The solution is to have groups of users rather than individual users as subjects that receive access rights from the authorization system. While several approaches to grouping users have been presented they are not powerful enough to describe a wide range of logical groupings. In this paper we develop a generalized approach to user group structures to solve these problems. We present structurings and primitives for user groups based on object-oriented concepts which are more powerful and general than those presented until now. Although formulated in the context of an object-oriented database system, our approach is general and could be applied to other data models, and even to operating systems.

Keywords: Access-matrix-based security, Authorization models, Data administration, Database security, Discretionary access control, Security of computer systems.

1 Introduction

Computer installations keep increasing in complexity: the lower cost of equipment and the tendency towards decentralization and distribution has increased the number of users that have some type of access to the computing system. In this scenario, the control of security becomes a more difficult problem and security administrators need ways to help their work. A large number of authorization rules is required when access rights are defined using access-matrix-based models. These rules are difficult to maintain and store and it is hard for a security administrator to understand the security implication of each rule.

Groups as accessing entities instead of individual users allow administrators to place users with similar functions in the same group and authorization rights can then be given with respect to the group. This improves the efficiency of the system because the number of authorization rules to be handled decreases dramatically. It also has the advantage of making possible the application of institutional policies in the definition of the groups; for example, all secretaries have similar functions and can be given a package of common rights. Because the number of distinct functions in a typical institution is not very large the total number of groups is reasonably low. Most authorization models include groups in one way or another. Groups are another application of the concept of *implied authorization* [9], where an authorization rule defined with respect to some composite structure (data or users) gives similar rights for/to each component.

We develop here a unified framework for groups. We first consider unstructured groups and formulate an authorization model based on object-oriented concepts. We also define a set of procedures to create, destroy, and manipulate user groups. We then extend this analysis to groups of groups and present an additional set of procedures. In particular, we propose three group structurings in analogy with the possible associations used in object-oriented modeling. We also show how to use these groups in evaluating access requests and to implement different security policies. Expressing the authorization system as a set of classes and associations in an object-oriented database allows the system to protect the authorization rules in the same way as the rest of the data in the database. If the database is of another type our approach is still useful as a design guideline.

Many authorization systems have used the concept of groups, e.g., [1], [2], [3], [5], [11], [12], [13], [17], [21]; our approach is more general in that all of these models can be shown to be special cases of ours. Our approach thus provides a unifying framework for all those models. Some of the most interesting earlier approaches are:

- The Inventory Control System (ICS), developed at IBM [12]. In their approach a group is an entity with which users and/or groups may be associated to access resources. They defined a set of special access types to handle groups, including *run* (use a resource in a group), *use*

(use resources and create files in the group), *control* (implies create and also allows a group member to connect other users to the group), and *join* (implies control and permits a user to define new system users and new subgroups). Four data structures: group, user, connect, and file, define the data needed to describe the group structure.

- SQL/Oracle. R. Baldwin proposed the concept of Named Protection Domains (NPD's), and applied it to manage security in SQL relational databases [2]. An NPD is a collection of application-oriented rights given to a set of users with similar functions. NPD's can be structured into trees of rights (the higher-level NPD's include the rights of the lower-level NPD's).

- The IRIS database system. This is an object-oriented database developed at Hewlett Packard [1]. Users are described by class *user* which includes operations to create and destroy user objects, return valid user names, and match passwords. A class *group* describes sets of users to which rights can be granted. Groups can be structured into trees similarly to the NPD's described above. There are also special operations for groups, e.g. to list the members of a group.

- Kelter did a systematic study of group structure in [13] and [14]. Additionally to the concept of tree-structured groups where supergroups include the rights of subgroups, he introduced the concept of specialized subgroups that inherit the rights from their supergroups.

- Bertino and Weigand [3] consider also inheritance from supergroups into their subgroups. They consider also the effect of inheriting content-dependent predicates (which we don't consider in this paper).

- Brüggemann [5] defines inheritance in subject groups and considers also the effect of negative authorization rules.

Some authorization systems for object-oriented databases also use data grouping to reduce the number of authorization rules, e.g. [5] and [15] use the granularity of the data as a criterion for implied access: access to a class implies access to all the instances of that class, etc. Other approaches, e.g. [7], take advantage of the semantic data structuring: access to a class implies similar access to its subclasses. These groupings are orthogonal to the user groupings considered here and could be combined with them for further reduction in the number of access rules. However, user groupings do not require any data grouping to be effective and can be used with any data model.

Several authors have proposed the concept of *user roles*, where a role is a group of users that has specific functions [2], [21]. We can consider roles as defining policies and groups as mechanisms

to apply these policies. Here we concentrate on mechanisms for group structuring regardless of their use as specific roles. In fact, our study is complementary to that type of studies.

Section 2 reviews basic concepts of object-oriented systems and of the authorization model. Section 3 discusses several issues related to groups. In Section 4 we propose policies and procedures to handle unstructured groups. Section 5 considers structures consisting of groups of groups. Section 6 considers access request validation while Section 7 provides a complete example of the application of these structures. Section 8 provides conclusions and directions for future work.

2 Background

2.1 Object-oriented concepts

There is a variety of models that use object-oriented concepts to specify a system design. They can be used as a semi-formal model that is relatively precise, that can be easily formalized, and can be used as a basis for implementation. To be concrete we adopt one of these here; the Object Modeling Technique (OMT) [19] which will be used throughout this paper. OMT is a methodology which consists of three submodels: The *object model* represents the static, structural, "data" aspects of a system. The *dynamic model* represents the temporal, behavioral, "control" aspects of a system. The *functional model* represents the transformational, "function" aspects of a system. A typical software application incorporates all three aspects: It uses data structures (object model), it sequences operations in time (dynamic model), and it transforms values (functional model). Each submodel contains references to entities in the other submodels.

The *object model* describes the structure of objects in a system — their identity, their relationships to other objects, their attributes, and their operations. Objects with the same data structure (attributes) and behavior (operations) are grouped into a *class*. A *class* is an abstraction that describes common properties of some entity of interest to an application. Each class describes a possible infinite set of individual objects. Each object is said to be an *instance* of its class. Each instance of the class has its own value for each attribute but shares the attribute names and operations with other instances of the class. An attribute of an object may take on a single value or a set of values [16]. An object may be an instance of only one class. An *operation* (method) is an action or transformation that an object performs or to which it is subject. Attributes and operations are referred collectively as *features*. Some features, denoted by $, are called *class features* and apply to all the objects of a class.

Classes with common properties can be *generalized* into superclasses which factor out these properties (*generalization association*). Conversely, subclasses can be said to be *particularizations*

of their superclasses. Generalization is denoted in this model by a small triangle ($\triangle$).

Inheritance is the property of subclasses where they share attributes and operations based on their hierarchical relationship. A class can be refined into successively more detailed *subclasses*. Each subclass incorporates, or *inherits*, all of the features of its *superclass* and adds its own unique features. A class may have any number of subclasses. A class may have any number of superclasses, and inherits attributes and operations from all of them; this is called *multiple inheritance.*

Aggregation is a form of relationship in which an aggregate object is *made of* components. The aggregate is semantically an extended object that is treated as a unit in many operations, although physically it is made of several parts. In diagrams this concept is represented by a rhomboid or diamond ($\diamond$).

A *relationship association* represents the fact that instances in different classes participate in common activities. For example, a student taking a course can be described in this way. *Multiplicity* specifies how many instances of one class may relate to a single instance of a related class and constrains the number of related objects. Multiplicity is often described as being "one" or "many", but more generally it is a (possibly infinite) subset of the non-negative integers. In diagrams a relationship is represented by a link at whose ends the corresponding multiplicities are defined (a black dot ($\bullet$) indicates "many".)

2.2 Security policies and authorization rules

In general, an authorization rule is a tuple (s, o, a, p, f), which defines that *subject* s has authorization of type a (access type) to those data values of security object o for which *predicate* p is true. Subject s can grant the access right (o, a) to other subjects if the *copy flag* f is true. Because these rules can represent most security policies this model has been used as a basis to describe many of the authorization systems for relational databases [9] and object-oriented databases [7].

If we are not concerned with the control of individual objects but with control at the class or attribute level we do not need to consider predicates. For those cases an authorization rule is just a triple (s, o, a) where s is a subject, a is an access type, and o is a class or a set of attributes. If we do not allow subjects to grant rights to other subjects we do not need a copy flag either. For space reasons we will not discuss here granting of rights.

As said earlier, in systems where subjects may be groups of users a right given to a group may be *implied* as a right for all the users in the group. We assume here that all the rights granted to a group are implicitly available to each group member.

Negative authorization rules are necessary to override implied access rights and are very useful to specify precisely the required authorization of some objects. For example, the system described

in [18] uses positive and negative authorizations: a subject may be denied access to an object either because it has no authorization for it or because it has a negative authorization on it. Negative authorization constraints are also required by the Orange book (that defines standards for the security of commercial systems) for security classes B3 and A1 [6]. Again for space reasons we will leave out the analysis of negative rights.

An important policy decision is the separation of *ownership* from *administration*. In the first case users own and administer their data; in the second case the information belongs to the enterprise, users are given access to it to perform their functions and special users (administrators) control the structure and the use of the information. Again, for conciseness we do not discuss administrative aspects, although clearly the user grouping operations would be used by administrators.

3 User Groups

As said earlier, user groups can be used for efficiency and as a way to define sets of rights based on the organization of an institution [9], [21]. The authorization rules take now the basic form (g, o, a, p), where g is a user group (remember that we left out the copy flag). However, these groupings bring the problem of how to interpret a given access request (since there is no direct mapping now from the components of the request to the components of the authorization rules). Users may belong to more than one group (although, in general, they will belong to only a few groups). Two or more groups may have access to the same objects (as well as other objects), i.e., they may share rights. In this case more than one access rule may be applicable to a given access request. For example given a request from user u, who belongs to user groups g_1 and g_2, for access of type a to object o, there may be two rules that apply:

$$r_1 : (g_1, o, a, p_1) \quad \text{and} \quad r_2 : (g_2, o, a, p_2)$$

Two basic policies to handle this case are [9]:

1. *The user chooses.* Some systems require a user to specify which group applies for a particular session. For example, a Multics user who works on several projects chooses one to provide the authorization context for a session.

2. *The rules are combined.* A user can receive the union of the rights of the groups to which he belongs. This allows, for example, an overall minimum level of rights to be specified for the Universal or Public user group (This group allows access to a basic set of objects). In our example, the request is valid if it is authorized by *either* r_1 or r_2.

Clearly, intersection the rules of the groups to which a user belongs is not reasonable; for example if she belongs to two disjoint groups she would get no access rights!

The two possibilities above may make sense in specific environments. For example, [11] makes a point for the use of the union of groups while [14] believes only one group should be active in a user interaction. Any mechanism for groups should be able to implement both of them.

Other issues that must be considered to define group structurings are:

- How are access rights associated? Are they associated only with groups, or could users also have individual rights? There is a tradeoff between complexity and flexibility in these two approaches. The simplest, and more uniform approach, is to associate rights only with groups (we can always define a group with only one user to accommodate special users) and we use this approach here.
- How to structure groups, i.e., is it possible to have groups of groups? As we discuss later (Section 5) this can significantly enhance the power of the model to describe different security policies.
- How the rights for groups are defined? This is an institution policy, ideally groups should correspond to user functions or roles and the group should receive the necessary rights for the required function to be performed (e.g., a mail clerk receives only enough rights to perform his job).
- How to evaluate access using groups, i.e., how to accelerate access request evaluation by taking advantage of group composition. We discuss this in Section 6.
- How to revoke granted rights; this is a more general problem that has been studied elsewhere [9]. It depends on the general problem of how are rights granted.
- How to find efficient implementation methods? While of practical importance, this is not discussed in this paper. Sandhu [20] considered this aspect in detail.

4 Policies for Unstructured Groups

We start our discussion with a system that uses only independent groups. We call these *unstructured groups*. We then consider groups that are related to one another, and we call this kind of groups *structured groups*. In this section we will discuss the first approach and develop a conceptual specification of its behavior, groups of groups are considered in Section 5.

We define a group as a set of users with common rights. In other words, access rights are defined for specific user groups, or inversely, a group can be seen as a set of rights. When we create a group, we are effectively creating a set of rights, i.e., usually groups are defined with respect to

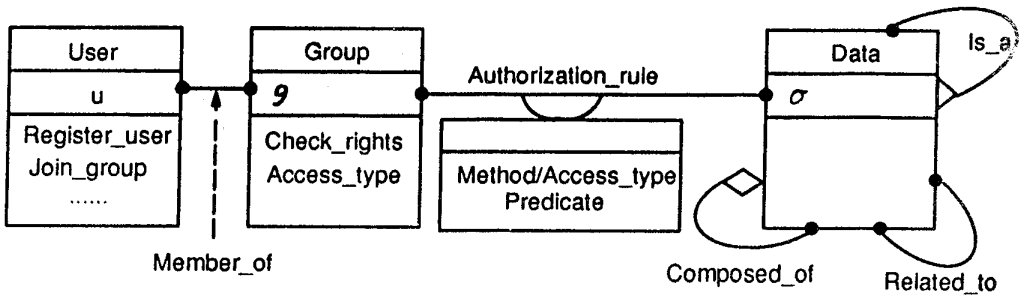


Figure 1: Class model for authorization rules

some functional task (role) that has to be performed and that requires access to a set of specific data items. This is a strict application of the need to know policy: a set of rights for each functional task. Formally, a group g is defined as: $g = \{u_i\}$, where u_i is any registered user of the system, $right(u_i)$ is the set of user u_i rights, and $right(g)$ is the set of group g rights. $right(u_i)$ and $right(g)$ are related as follows:

$$(g, o, a) \in right(g) \wedge u_i \in g \rightarrow (u_i, o, a) \in right(u_i)$$

We do not define a *universal* or *public group*, to which each user belongs by default and which provides some basic access rights, but require that any right be explicitly given. A user may belong to one or more groups according to his functional tasks (job assignment). All rights are associated with groups, i.e. users only acquire rights by belonging to some group. A new group can be created for a single user if there is no group that accommodates her functions. If we also apply the policy of separation of use from administration, groups can be operational groups, database administrator groups and security administrator groups. However, as said earlier, that distinction is not fundamental for our group development and we do not pursue it further here (see [7] for further discussion).

As shown in Figure 1, authorization rules can be represented using OMT as a relationship between *group* and *data*. The "data" class shows the possible structurings of the data in an object-oriented database. Class User represents all the registered users with id u. Two basic operations are shown in this class: Register_user and Join_group with obvious meaning. Other useful operations would be Delete_user, Leave_group, $Display_users, etc. [10]. The relationship Member_of describes which users are in what groups. Class Group represents the groups in the system, for which relationship Authorization_rule defines the corresponding rights. The relationship

attribute *Method/Access type* defines the operator that the user is authorized to apply to the data while *Predicate* indicates the corresponding data-dependent restriction. In some systems access is controlled at the read/write record level [7], [15]. In others, at the method level [8]. The method *Check_rights* evaluates if a given request is authorized for some subject. *Check_rights* could also be attached to data if we think that its invocation is the result of accessing some specific data entity. Method *Access_type* returns the method authorized to a user for a given class. As indicated earlier, the hierarchical structure of classes and subclasses may also be used to define *implied accesses* [7], thus further reducing the number of explicit rules.

An example of a class definition is given below. (Remember $ denotes a class operator as opposed to an object operator.) Some of these methods may not be needed in specific implementations or additional methods may be needed; that is, this is only a typical definition.

Class GROUP **is**
  g: **string**; - - group identifier
  proc Create_group (g); - - Creates a new group g.
  proc Delete_group (g); - - Deletes an existing group g.
  proc Divide_group ($g, g(1), ..., g(n)$); - - Divides one group into n groups.
  proc Combine_group ($newg, g(1), ..., g(n)$); - - Combines n groups into one group.
  proc Add_right (g, o, a); - - Adds access right (o, a) to group g.
  proc Delete_right (g, o, a); - - Removes access right (o, a) from group g.
  proc Display_G_attributes (g); - - Lists the attributes of a group.
  proc Check_rights($g, o.a$); - - Checks if a group is authorized to apply a to data o.
  proc Access_type(g, o, a); - - Returns the value of a for a given group with respect to data o.
  proc Add_member(g, u); - - Adds user u to group g.
  proc List_rights(g); – lists all the rights of a given group.
  proc $Display_groups; - - Lists all the groups in the system.
  func $Is_a_group ($g$): **Bool**; - - Checks if a specific group exists in the system.
  func $Has_a_right ($g, o, a$): **Bool**; - - Checks if a group has a specific explicit access right.
  proc $Display_member; - - Displays all the (user, group) pairs.
end

Specifications for the operations of these classes are given in detail in [10]. We show here two cases as illustration.

proc Join_group (u, g);
- - Adds a user to a group.
begin
  if not Is_a_group (g)
    then return error – group does not exist
    else if User.Is_a_member (u, g)
      then return error – user u is already a member of g
      else begin
        Add_member(g, u)

```
                    end if
end Join_group

proc Add_right (g, o, a);
- - An access right tuple (o, a) is added to a group rights package
begin
      if not Is_a_group (g)
        then return error
        else if Has_a_right (g, o, a)
                then return error - - group already has this right.
                else Authorization_rule := Authorization_rule ∪ (g, o, a)
end Add_right
```

5 Policies for Structured Groups

5.1 Group structuring

In an institution there exist more complex structures than just groups. For example, a set of departments can become a division and be under the direction of a single manager. Sometimes, for work reasons it is necessary to collect small groups of employees to work on specific projects. This creates the need to have groups of groups. We present now three types of structurings and some of the corresponding procedures to manipulate them.

As seen in Section 2 there are three basic ways to associate classes in the object-oriented model [19], namely generalization, aggregation, and relationship. By analogy these three associations can be used as a basis to structure groups of users and all of them can be given a useful meaning for representing the institution organization:

- *A generalization structure* describes structures where the subgroups perform more specialized operations than the supergroups. For example, Figure 2 shows a group of programmers (Programmer) which can be specialized to programmers that perform more specific tasks, e.g., System Programmer, Real-time Programmer, etc.
- *A composition structure* describes structures that represent the administrative or physical division of people. For example Figure 3 shows a company divided into three groups of employees that belong to different departments.
- *A relationship structure* describes associations between groups needed to perform new jobs. It effectively describes a new group formed taking people from two (or more) existing groups. For example in Figure 4 some real-time programmers and some processor designers are

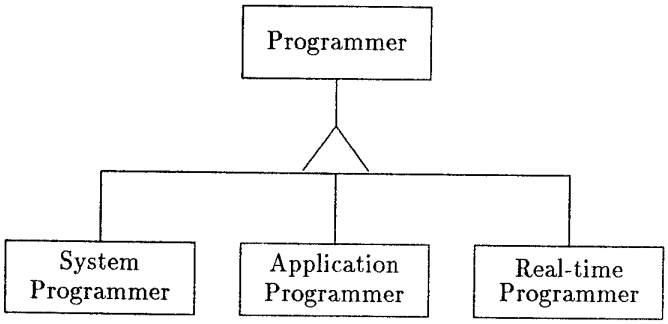


Figure 2: Generalization structure

assigned to work in a project called Real-Time Computer defined by relationship Real-time Computer. In forming this type of group we ignore the correspondence between set elements indicated by a relationship, i.e., we only consider the presence of the elements themselves. We also describe this group by the name of the relationship; a more consistent representation from a notational viewpoint would describe this relationship as another class [10], but we have not done this.

A specific user may belong to different groups, for example, an individual could be in the group of application programmers and in the group Manufacturing. In general, these groupings can represent the permanent jobs of the users as well as temporary assignments. Sometimes both types of groups may coincide, for example a company may divide its departments according to specialty; in this case, in the example of Figure 2, we would have also a composition structure. As another example, Figure 5 shows a group New_systems which is made up of three groups: New_op_sys, New_user_if(interface), and New_real_time_comp, where each component group is a relationship group combining designers of different specialities. Here New_systems would be the group including all designers working in new systems development.

The principle of "need to know" is fundamental to design a secure system [9]. This principle establishes that users should be given just enough rights to perform their duties. Groups can reflect the structure of the functional tasks to be performed and the policies of the institution. If they are the only way to acquire rights, by controlling access to groups it is possible to enforce institution

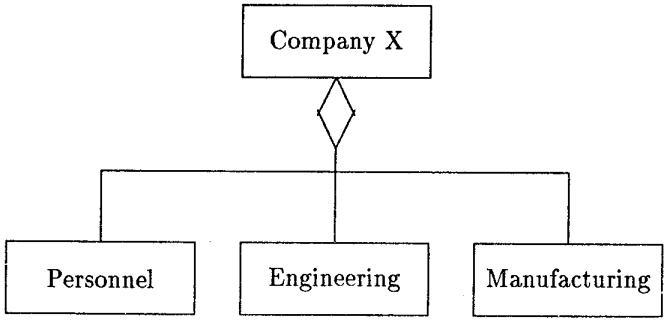


Figure 3: Composition structure

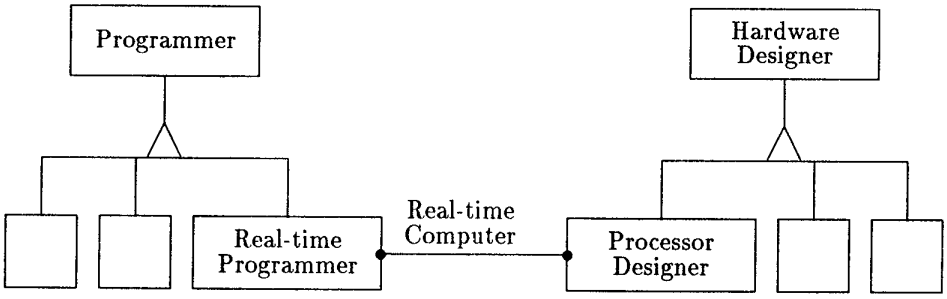


Figure 4: Relationship structure

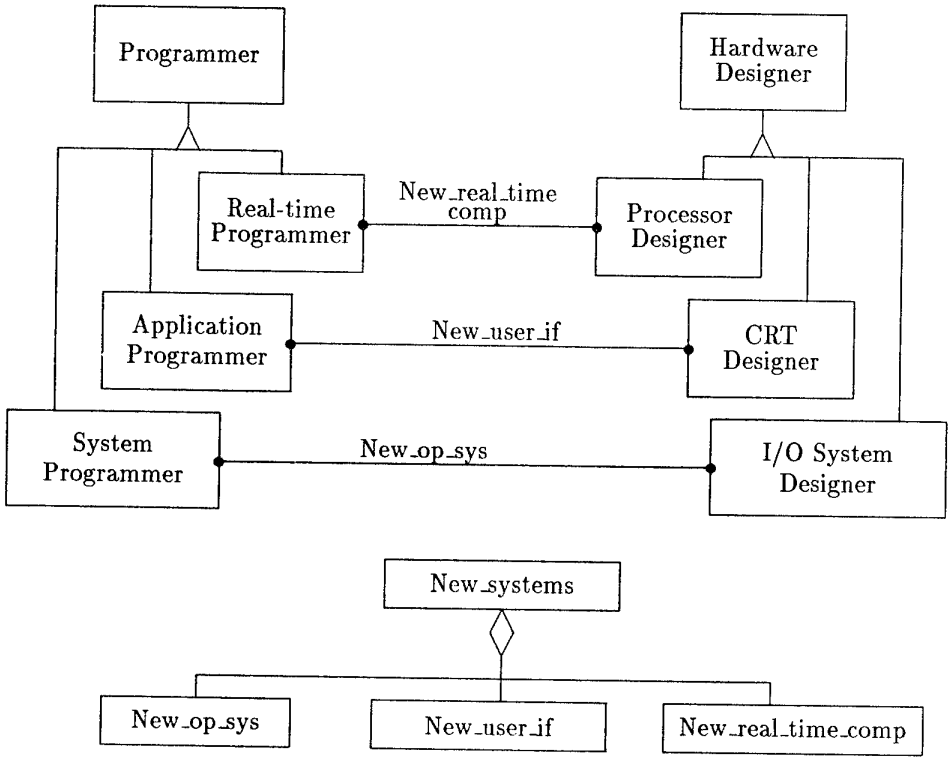


Figure 5: Group combinations.

security policies in a simple way. Consequently we can define the following group policies (GP):

- *Policy* GP_1: Users in subgroups of generalization structures inherit all the rights from their supergroups.
- *Policy* GP_2: Users in supergroups of composition structures acquire all the rights of their subgroups.
- *Policy* GP_3: Users in relationship structures bring to the new group their own group rights and acquire the specific rights implied by the needs of the new group.

We use the following notation: $right_e$ is a set of explicit rights, i.e. rights that are explicitly associated with the group under consideration; $right_i$ is a set of implicit rights, i.e. rights that can be derived from the explicit rights using one of the above three policies. $G = \{g_1, g_2, ..., g_k\}$ is a generalization of k groups, $C = \{g_1, g_2, ..., g_k\}$ is a composition of k groups, $R = \{g_1, g_2, ..., g_k\}$ is a relationship group among k groups. A group g may have explicit or implicit rights, i.e.:

$$right(g) = right_e(g) \cup right_i(g)$$

We express these policies formally as:

$$GP_1: \quad g_i \in G \wedge r \in right(G) \rightarrow r \in right_i(g)$$

$$GP_2: \quad g_i \in C \wedge r \in right(g) \rightarrow r \in right_i(C)$$

$$GP_3: \quad u_i \in R \wedge r \in right(R) \rightarrow r \in right_i(u_i)$$

Note that a relationship group is not strictly a group of groups as the generalization and composition groups. We can visualize a relationship group as a group formed taking specific users from two or more existing groups and collecting them into a new group.

Policy GP_1 is justified because subgroups define more specialized groups. In this case the members of the subgroups should have the rights of their supergroups and in addition the rights needed to perform more specialized functions. For example, a Real-time Programmer is a programmer and should possess all the rights needed by programmers to perform their functions. Additionally, a Real-time Programmer needs access to specialized tools.

Policy GP_2 makes sense to describe data access. For example the group that develops programs for the whole company should have access to the programs developed by the individual departments. Another justification comes from seeing the subgroups as the result of dividing an existing group; in this case the large group is a way to refer to the set of subgroups and includes all of their functions and consequently all of their rights. This policy describes the hierarchical structure of most institutions where a high-level group has more power and rights than a lower-level group.

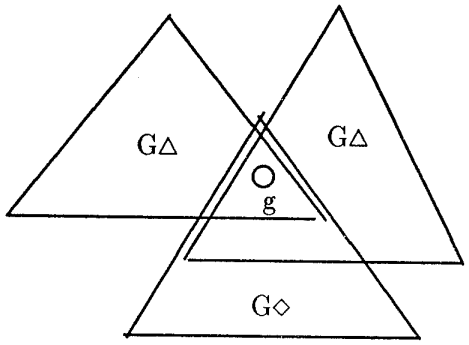


Figure 6: Group of groups graphs.

Finally, Policy GP_3 would be useful to describe projects which require the combination of people with different specialties or from diverse departments. Because their function is specific the members of this type of group should have project-specific rights in addition to their other group rights based on their specialty or the department to which they belong.

As we said earlier, groups are very important for efficiency because there is no need to write individual authorization rules defining the rights of each user; if a user can join a given group because of her functions she automatically acquires all the group rights. Groups of groups allow to reduce the number of explicit rules even further by taking advantage of the inherent hierarchies present in most institutions. Negative authorization rules and predicates can be used to provide a more precise control of access (although we do not show that in this paper). Section 7 shows a detailed example of the application of these policies.

5.2 Procedures

We add now a set of procedures to manipulate groups of groups. All these procedures assume that the structure of a group of groups is described by a class GROUP_OF_GROUPS which includes the interconnection graph of the groups. We also assume that each group G in GROUP_OF_GROUPS can include only one type of association, i.e., G can be a generalization, aggregation, or relationship group of groups. This is not a restriction since we can decompose a complex group into homogeneous subgroups. Notice also that a given group g can belong to any number of groups G of any type. Figure 6 shows this situation; here G◇ represents an aggregation graph and so on. In particular, generalization and aggregation graphs are trees; relationship class graphs can be seen as 2-level trees if the relationships are binary or ternary. The class representation of group of groups could be as shown below.

class GROUP_OF_GROUPS **is**
G: **string**; – identifier for group of groups graph
Children ($g \in G$): - - For each group g in the group of groups this data structure describes its children.
Type: {generalization, aggregation, relationship}; –The three types of group graphs
proc Create_group_of_groups (G, t) - - Creates an empty group of groups G with type t.
proc Delete_group (G, g) - - Deletes group g from G.
proc Add_group (G, g_1, g_2)
- - add group g_1 as the child of g_2 in G, the type of G determines the type of association.
proc Ancestor (G, g) - - Finds the ancestors of a group g within a group G.
proc Descendant (G, g) - - Finds the descendants of a group g within a group G.
proc Related_group (g) - - Finds the related groups of a group g in the system.
end

As before, the details of these procedures can be found in [10].

6 Evaluating Authorization

Access requests from user programs or query languages must be compared with the authorization rules to decide if the requested access is legal. This is normally performed by some evaluation algorithm [9]. A request has the general form (s′, o′, a′), where s′ is the requesting subject, o′ is the requested data item, and a′ the intended access type. Since authorization rules have groups as subjects if we start from a request from user u, one must determine first the effective subject, G_{eff}, considering the groups to which he might belong. The data item can be a class or an attribute of a class. The access type may be a method or read/write actions on attributes. In other words, we are trying to match up ($g' \in G_{eff}$, o', a') against an authorization rule of the form (g, o, a) explicit or implicit.

The evaluation algorithm is just one of the methods (Check_rights) in the authorization model of Figure 1. To find the effective subject we need to determine all the inherited and acquired rights according to the group structuring (Figure 7). A high level expression of the evaluation of a request is shown below The algorithm accommodates any policy with respect to active groups by adjusting the function Group_membership to return either only one group or a set of groups:

func Check_rights (s′, o′, a′): **Bool**;
- - s′ comes from login, o′ and a′ are defined from the application language interface.
- - G_{eff} is the effective group of s′ which defines the subject of the authorization rule
begin
G := USER.Group_membership (s′);
- - If we use the policy of OR-ing all the groups to which a user belongs, method Group_membership
- - returns of set of all the groups to which a user belongs. If the user can only use one of his groups the
- - user would be allowed to select one of the groups in this set. This would imply to add here an

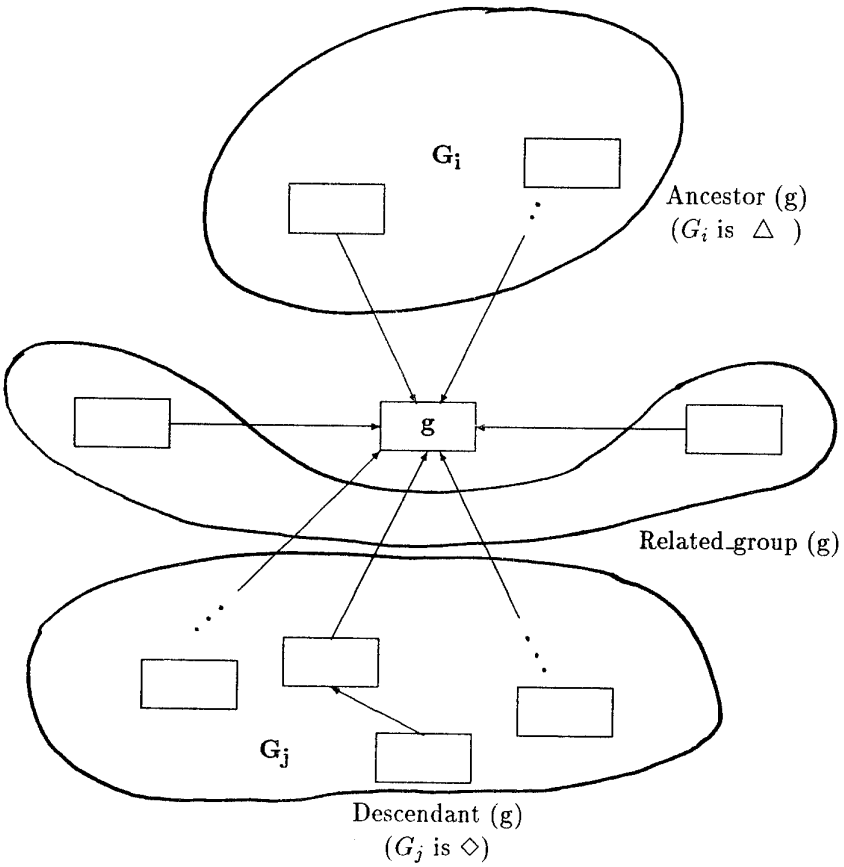


Figure 7: Determination of effective subject.

```
- - interactive step to receive the user's choice.
if G ≠ φ
  then begin
    G_anc := ∪_{g∈G} Ancestor (g);
      - - This finds all the ancestors of g in all generalization group graphs to which g belongs.
    G_desc := ∪_{g∈G} Descendant (g);
      - - Finds all the descendants of g in all aggregation group graphs to which g belongs.
    G_rel := ∪_{g∈G} Related_group(G, g);
      - - This finds all groups directly related to g.
    G_eff := g ∪ G_anc ∪ G_desc ∪ G_rel;
      - - G_eff is used as subject for possible access rules that authorize the request
    if (o', a') ∈ (o, a) of {set of rules with g ∈ G_eff as subject}
      then return True
      else return False
    end if
  else return False
 end if
end
```

This algorithm works as follows:

- First the effective subject is determined by considering all the direct, inherited, and acquired rights for subject s' (Figure 7).
- We have now a set of groups that are the possible subjects for access rules authorizing this request, let these be $G_{eff} = \{g_1, g_2, g_3\}$.
- We can think of the relationship *Rights* as a table linking subjects (groups) to data. We then search this table and determine the rules that have g_1, g_2, or g_3 as subjects. If any of those rules has o = o′ as security object we check in the relationship table the corresponding access type a. If this matches a' the request is authorized.

One should note here that this approach applies to any type of class structuring for the protected data. In the model of [7] the set of rules that have G_{eff} as subject are determined by implication along the data hierarchy (implied accesses are inherited from superclasses or subclasses) while in other models they would be determined in other ways. For example, in systems without implied accesses one needs to find a rule that matches exactly the requested data unit, i.e., $o = o'$. Another important issue is the meaning of G; it could be interpreted as the set of all the groups to which u′ belongs (as shown in the algorithm) or as one of these selected by the user [13] (See discussion in Section 3). The algorithm is also general in this sense, in a specific implementation one could adopt either policy. For example, in certain applications the structure of a program under development can be modeled very conveniently by a class hierarchy describing its components. Versions of a

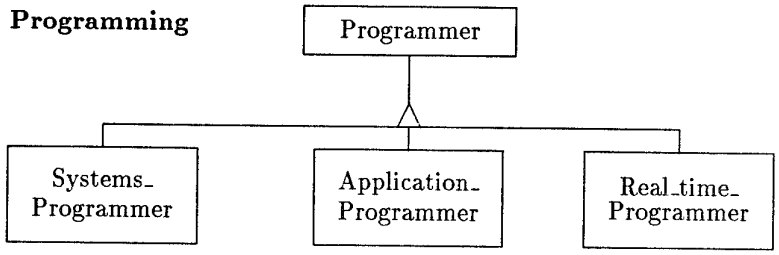


Figure 8: A grouping of programmers.

program can be described by relationship associations between programs. This approach provides then a unified view of the complete system, as well as an easily implementable design.

Two aspects must be considered in the implementation of this algorithm:

- *Complexity.* In general the groups of groups for a real application will be simple, with at most three to four levels. This implies that the propagation of access rights is rather short. To improve efficiency even more the effective subject can be determined at login time or ven at compile time.
- *Graph structure.* If groups are defined without control, loops may occur. Because of the recursive nature of the algorithm these must be avoided. The Add_group method should check for possible loops.

7 Application of the proposed approach

We will show the practical value of our proposal through a detailed application example. We consider a software development environment because it requires a rich variety of authorization policies [4].

We assume a typical software development company and we see how logical actions of its operation can be implemented by specific commands from the set of group primitives described above.

1. Assume the initial state of the development system is described by the graph of Figure 8. Note that Programming is the groups of groups' name, that is, up to this point the operation of the system is such that only one grouping with two levels is necessary, i.e., we only have programmers and they are classified in several types requiring different types of tools.

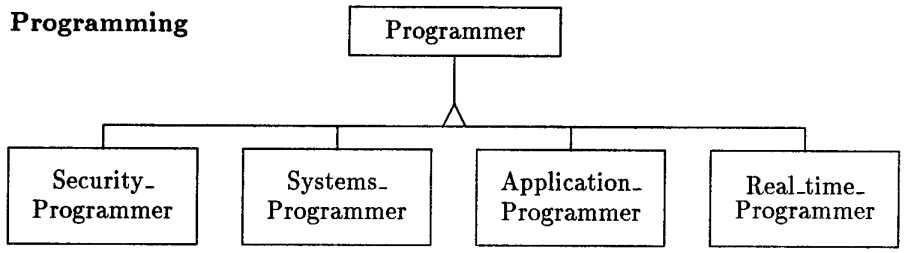


Figure 9: The company after step 3.

Programmers have access to standard compilers and editors, while real-time programmers for example, can also access real-time simulators, a Petri-Net specification tool, and other specialized tools.

2. Mary Jones (an application programmer) and Eduardo Lu (a systems and real-time programmer) join the company and are assigned to their corresponding groups through the commands:

   Join_group (M. Jones, Application_Programmer)

   Join_group (E. Lu, Systems_Programmer)

   Join_group (E. Lu, Real-time_Programmer)

   Now, for example, M. Jones can use the specialized tools needed by application programmers. She inherits rights to access the standard tools needed by all programmers as well.

3. Now the company decides to go into the high-security systems market and they hire several programmers that are specialists in secure systems. In order to incorporate these specialists in the system we perform the following actions:

   a) Create a new group of security programmers:
   Create_group (Security_Programmer)

   b) Add to this group the new k programmers just hired:
   Join_group (name_1, Security_Programmer)
   .....
   Join_group (name_k, Security_Programmer)

   c) Make this group a generalization subgroup of Programmer:
   Add_group(Programming, Security_Programmer, Programmer)

   Now the company structure looks as shown in Figure 9.

4. The market for applications programming is poor and the company decides to abandon this field. The members of this group will be reassigned or laid off.

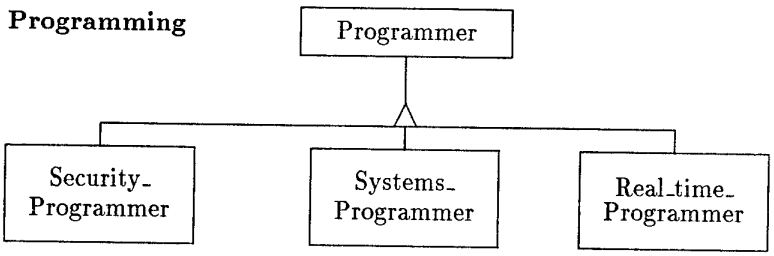


Figure 10: The company after step 4.

a) Group Application_Programmers is deleted:
Delete_group (Programming, Application_Programmer)

b) Some Application Programmers, e.g., R. Johnson, are laid off:
Leave_group (R. Johnson, Application_Programmer)
R. Johnson is also deleted from class User.

c) Others are just reassigned:
Join_group (M. Jones, Systems_Programmer)

The company now looks as shown in Figure 10.

5. Management realizes that it is impossible to develop good systems without hardware specialists and hires several of these. They are assigned to three specialization groups: processor designers, I/O system designers, and CRT designers. The new state of the company is shown in Figure 11.

6. This organization works reasonably well because the users have the rights they need to use their tools and the systems they are developing. It assumes that a project is divided into the different specialties according to their abilities. If there is only one project under development that is all we need. However, we want more flexibility. What if we need to select *some* software and hardware designers to work in a high-priority project, e.g., a new real-time computer? This is the reason for the relationship groups that we have introduced. Note that an aggregation group will not do: an aggregation group including for example the group Real-time_Programmer and Processor_Designer will select *all* of these people not just a specific set of them. We define then a relationship group: Relate_group (New_real_t_comp, Real-time_Programmer, Processor_Designer). We then assign rights to this group according to its intended function. Then we can indicate its members: Join_group (E. Lu, New_real-t_comp) etc. The new state of the company at this stage is shown in Figure 12.

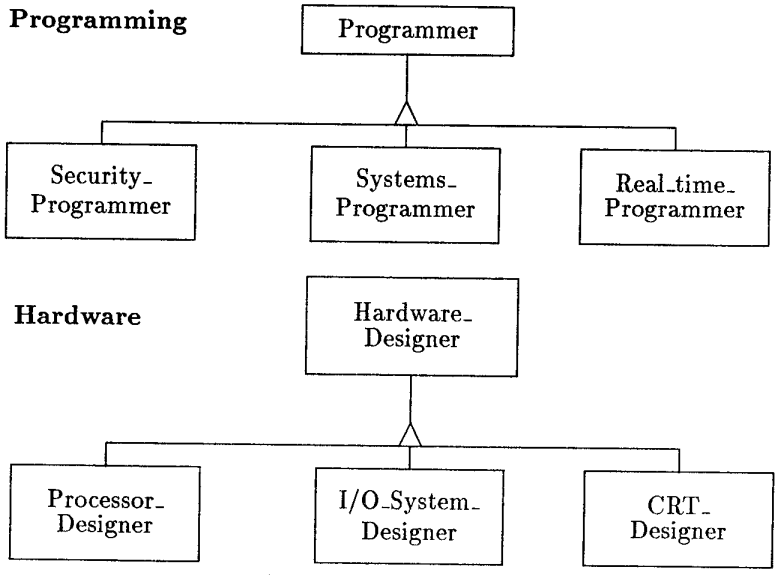


Figure 11: The company after step 5.

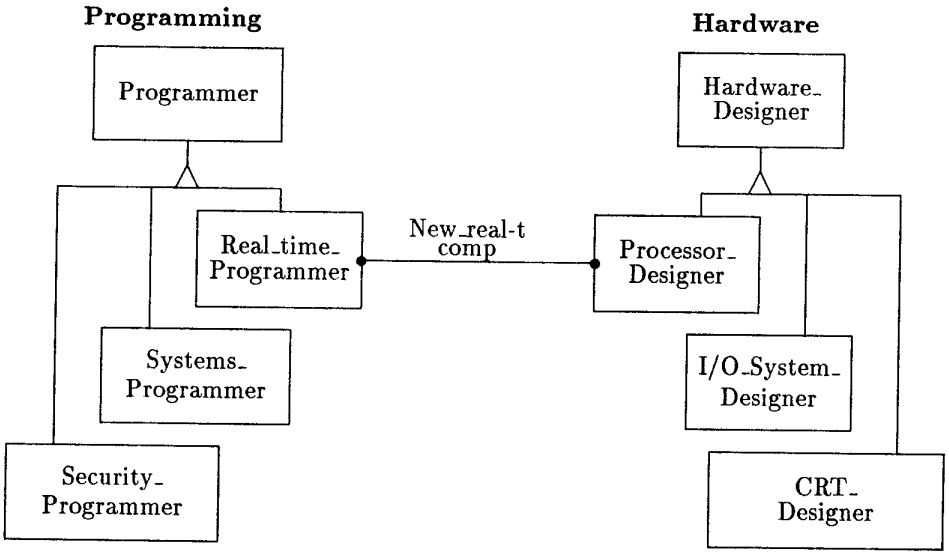


Figure 12: The company after step 6.

8 Conclusions

The main contributions of this work are:

- A basis for unstructured groups of users including a set of procedures to manipulate them (a more detailed set of procedures can be found in [10]).
- A framework for structured groups of groups including the corresponding procedures for their handling. A detailed comparison with other approaches can be found in [10].
- A new concept to structure groups of groups, the relationship group, which had not been proposed in any previous system and which is useful to define some types of security policies, i.e., it enhances the precision of the authorization system.
- A new set of authorization policies for groups of groups which can reflect the least privilege policy into the institution group structure.
- The formulation of the authorization system itself in terms of classes and associations between these classes. This provides unity to the complete system design in that all of these models can be shown to be special cases of ours. Our approach also provides a direct basis for implementation. More importantly, it allows the authorization system to protect itself. This had not been done for object-oriented databases; in fact other models, e.g., [13], use low-level primitives for this purpose. This also results in simpler algorithms in the authorization system.
- The application of the proposed group structures to the evaluation of access requests. This can significantly improve the efficiency of the authorization system. A specification of the necessary algorithm is presented. This algorithm can be combined with evaluation algorithms based on data structuring [7].

In fact, although all these concepts were developed with an object-oriented database in mind, they can be applied to other types of databases. Further, they can also be applied to operating systems. They are particularly useful for distributed environments where there are many users with similar roles.

References

[1] R. Ahad, J. Davis, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuegbe. "Supporting access control in an object-oriented database language". In *Proceedings of the European Conference on Extending Database Technology, EDBT '92*, Vienna, Austria, March 1992.

[2] R. W. Baldwin. "Naming and grouping privileges to simplify security management databases". In *Proc. IEEE Int. Symp. on Research on Security and Privacy*, pages 116–132, 1990.

[3] E. Bertino and H. Weigand. "An approach to authorization modeling in object-oriented database systems ". To appear.

[4] A. W. Brown and J. A. McDermid. "Learning from IPSE's mistakes". *IEEE Software*, 9(2):23–88, March 1992.

[5] H. H. Bruggemann. "Rights in an object-oriented environment". *Database Security V: Status and Prospectus.* C. E. Landwehr and S. Jajodia (Eds.). Elsevier Science Publ. B.V., pages 99–115, 1992.

[6] Department of Defense Computer Security Center, editor. *"Trusted Computer System Evaluation Criteria".* DoD 5200.28-STD. Dept. of Defense, December 1985.

[7] E. B. Fernandez, E. Gudes, and H. Song. "A model for evaluation and administration of security in object-oriented databases". *IEEE Trans. on Knowledge and Data Engineering*, 6(2):275–292, April 1994.

[8] E. B. Fernandez, M. M. Larrondo-Petrie, and E. Gudes. "A method-based authorization model for object-oriented databases". In *Proc. of the OOPSLA 1993 Workshop on Security in Object-Oriented Systems*, pages 70–79.

[9] E. B. Fernandez, R. C. Summers, and C. Wood. *Database security and integrity.* System Programming Series. Addison-Wesley, Reading, Massachusetts, 1981.

[10] M. H. Fernandez. "Group structures in object-oriented database authorization". MS Thesis, Dept. of Computer Sci. & Eng., Florida Atlantic University, 1992.

[11] R. Gagliardi, G. Lapis, and B. Lindsay. "A flexible and efficient database authorization facility". RJ 6826 (65360), IBM Research Division, Yorktown Heights, New York, May 1989.

[12] H. M. Gladney, E. L. Worley, and J. J. Myers. "An access control mechanism for computing resources". *IBM Sys. J.*, 14(3):212–228, 1975.

[13] U. Kelter. "Group-oriented discretionary access controls for distributed structurally object-oriented database systems". In *Proc. European Symp. on Research in Comp. Security*, pages 23–33, 1990.

[14] U. Kelter. "Discretionary access controls in a high-performance object management system". In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 288–299, May 1991.

[15] W. Kim. *Introduction to object-oriented databases.* MIT Press, Cambridge, MA, 1990.

[16] W. Kim. "Object-oriented databases: Definition and research directions". *IEEE Trans. on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[17] J. D. Moffett and M. S. Sloman. Domino Paper Arch/IC/3, Imperial College of Science, Technology, and Medicine, London, UK, 1991.

[18] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. "A model of authorization for next-generation database systems". *ACM Trans. on Database Systs.*, 16(1):88–131, March 1991.

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F.Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall, Englewood Cliffs, N. J., 1991.

[20] R. S. Sandhu. "The ntree: A two dimension partial order for protection groups". *ACM Trans. on Comp. Systs.*, 6(2):196–220, May 1988.

[21] T.C. Ting. A user-role based security approach. *Database Security: Status and Prospects.* C. Landwehr (Ed.). North-Holland, pages 187–208, 1988.

User-Role Based Security in the ADAM Object-Oriented Design and Analyses Environment

M.-Y. Hu
IBM Corporation
1311 Mamaroneck Ave
White Plains, NY 10605
myhu@vnet.IBM.COM

S. A. Demurjian and T. C. Ting
Computer Science and Engineering Department
University of Connecticut
Storrs, Connecticut 06269-3155
steve@brc.uconn.edu, ting@eng2.uconn.edu

Abstract

Over the past three years, our work has explored the attainment of user-role based security (URBS) for discretionary access control (DAC) within an object-oriented design model. Our approach has extended the public interface (which defines the means for accessing classes or object types) to allow its methods to be selectively assignable (or prohibited) on a role-by-role basis. This allows different users at different times to have particular access to the public interface based on their specific roles.

The work presented in this paper details our prototyping efforts as we transition from research on URBS and DAC to the object-oriented design and analyses environment ADAM. ADAM, short for Active Design and Analyses Modeling, is a language-independent environment that automatically generates compilable code in C++, Ontos C++, Ada83, or Ada9X (object-oriented extension to Ada) from designs that have been supplied via text and form-based input. ADAM unifies the structural and security requirements, elevating the latter to a first-class citizen in the design process. ADAM also offers a framework of analysis techniques that are intended to support a more precise and accurate characterization of an application and its security requirements. Note that we also present recent research results on the support for integrity constraints within our model and ADAM, and discuss their potential impact on security considerations.

1 Introduction

Over the past three years, our research emphasis has concentrated on the investigation and attainment of discretionary access control (DAC) within an environment that supports object-oriented design and offers user-role based security (URBS) [2,10,19] as an

equal partner in the development process. URBS was proposed in 1988 [13,18] as a technique which focuses on individuals by characterizing and defining their responsibilities within the application as the means for identifying and establishing security privileges. We have chosen the object-oriented paradigm, since it has drawn much interest in recent years, in academia, industry, and government, and appears to offer unique capabilities that promote the design process (e.g., encapsulation and hiding - public interface vs. private implementation) and facilitate software evolution (e.g., representation independence and inheritance). Existing support for DAC in the object-oriented approach is limited to the public interface (i.e., the set of all visible methods, their parameters, and their return types for each object type or class), where all users, regardless of their needs within the application, have full access to all methods in the public interface.

Our research efforts have sought to customize access to the public interface, to allow different individuals to have particular access to specific subsets of the public interface at different times via their roles within the application. Our approach [2,10,19] has focused on establishing privileges by assigning (positive) and prohibiting (negative) methods based on roles. A role *assigned* a method can invoke the method, and by inference can also access instances on the object type on which the method is defined, and potentially read and modify private data that the method might use. A *prohibited* method restricts access in a similar fashion. The idea of defining both assigned and prohibited methods is important, since it allows the possible problem of information leakage [16] due to the inheritance between object types to be addressed.

There have been a number of other efforts related to our work. The approach in [14] is similar to our process of method assignment, but differs since they assign objects and authorization types to roles. Our approach also contrasts to [13], where the access rights/permitted roles are assigned based on data levels. Our approach and [15] both have the goal of providing different interfaces to different users, but differ since they assign views based on data. When considering negative privileges, our concept of prohibited methods is similar to the concepts of negative authorization in [14], denied roles in [13], and permission tags in [1], but differs since all of their efforts emphasize data; we focus on object types/methods. Other efforts for security in object-oriented systems via mandatory access control [12,17] differ from our DAC approach.

Our purpose in this paper is to report on the status of our prototyping efforts for supporting object-oriented design that includes URBS definition. Ongoing work has resulted in the development of an object-oriented environment, ADAM (short for Active Design and Analyses Modeling), that is capable of supporting both the design process and generating compilable code in multiple languages [6,7,8]. ADAM currently supports code generation for two dialects of C++ (GNU C++ and Ontos C++ - an object-oriented database system), Ada1983 [4], and Ada9X [5]. This paper reports on the incorporation of a portion of our previous security research into the ADAM environment, focusing on the definition of security privileges on user roles [10].

Our approach also contains a framework of analysis techniques [2,10,19] that operate from two perspectives to indicate: which user roles have access to a specific aspect (OT, method, private data) of an application; and, what is the access to the application of a chosen user role. Analyses are supported in two different ways within ADAM. First,

as privileges are being defined for different user roles, they are automatically checked against all existing privileges for that role to identify conflicts or inconsistencies in real-time. Second, at any time during the overall process, a security designer can initiate a large and varied number of analyses that allow him(her) to understand and evaluate the defined privileges against the desired security requirements of the application. These analyses can be used by the designer to investigate realized privileges against the application's intended security requirements. This paper will discuss the ADAM prototype, with an emphasis on its support for security definition and analyses.

The remainder of this paper is organized into four sections. In Section 2, we describe the object-oriented model and URBS definition capabilities of ADAM, using a health care application (HCA). Section 3 reviews the available analysis framework that allows the designer to understand and evaluate the application from complementary perspectives, with the goal to support a more precise characterization of an application. In Section 4, we detail ongoing research and prototyping issues, including an extension to the object-oriented model and ADAM for supporting integrity constraints, and a discussion of next-step research concepts for security enforcement. Finally, Section 5 summarizes the paper and indicates future plans.

2 Object-Oriented Design, URBS, and ADAM

The object-oriented design model for ADAM is tightly integrated into the environment, with the semantic, scope, content, and context of each modeling construct clearly defined. There is no specific syntax for the design model; choices are made via menus, browsers, etc., and text is directly entered by the designer using forms. Thus, the environment stresses language independence by focusing on design and allowing code to be generated in a variety of target languages for supporting a transition to the implementation effort. ADAM supports incremental design by allowing design data to be stored persistently in the Ontos database system. Design-analyses are promoted through *profiles* [8,10], which are detailed requirements on the semantic content and context for all constructs of the application. Profiles have two purposes:

1. Force software engineers to supply detailed information as an application is designed.
2. Provide on-demand and automatic analyses for feedback to software engineers whenever an action in the environment results in a conflict or possible inconsistency.

To support the entire design, development, and analyses processes, the ADAM environment has been partitioned into two stages: design phases and semantic perspectives.

Design phases (DPs) are intended to be used for constructing the application structure and behavior by segmenting the design process into logical parts. That is, through DPs, software engineers can define, modify, and evolve applications, including both its information and security characteristics. Throughout the design process, profiles are entered by software engineers and, in some cases, this data is propagated throughout

the entire environment. As software engineers develop and detail their designs, DPs also offer automatic feedback as discussed above. At any point during the design process, code can be generated so that designers may inspect whether generated code meets their needs and requirements.

Semantic perspectives (SPs) are the stage in ADAM that can be used by software engineers to analyze structure and behavior by examining different context views of an application's content. By understanding the semantic associations through feedback based on information, methods, object types, and user roles, software engineers can refine and redefine their designs. The analyses supported via SPs are accomplished using profiles. However, the SPs are a "read-only" world; to correct problems, changes must be made in the DPs.

The current implementation of ADAM (as of July 1, 1994), supports five design phases (three for modeling constructs and two for URBS and authorization) and two semantic perspectives (for object types and URBS). The remainder of this section discusses four of the five design phases and in the process reviews the major concepts using examples from a health care application (HCA) [10]. Note that ADAM has been implemented on a Sun architecture under a Unix environment using X windows, InterViews 3.01, and AT&T C++ 2.0.

2.1 Object-Type Specification

In the object-type-specification phase of ADAM, the designer can define object types (OTs), attributes, and methods through associated profiles. This work has been explored elsewhere [8], and is only briefly reviewed. An *attribute profile (AP)* includes:

1. a prose *attribute description* on the purpose of the attribute in the OT
2. the name and type of the attribute
3. a list of the methods that access the attribute and an indication of whether the attribute is read and/or written by each method

A *method profile (MP)* includes the following information:

1. a prose *method description* for the method's actions within the application
2. the method's name, return type, and parameter list (with names/types)
3. the read/write set for each private attribute used by the method M_i
4. the other methods that are called by M_i to accomplish its task

An *OT profile (OTP)* consists of following information:

1. a prose *OT-description* for the purpose of the OT in the application
2. the OT's name
3. the persistency status of the OT
4. the attribute profiles for all private attributes
5. the method profiles for all methods

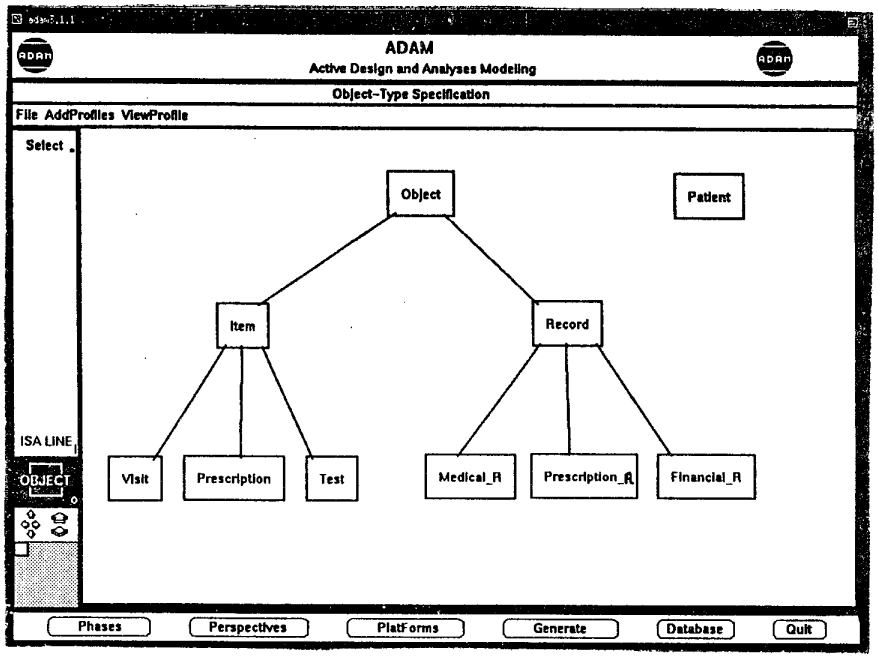


Figure 1: Object Types of a Health Care Application (HCA).

6. the relationships that involve the OT
7. the supertypes and subtypes of the OT

A subset of OTs of a HCA, based on prior work [10], is shown in Figure 1. In this figure, there are ten OTs: `Object`, `Item`, `Record`, `Visit`, `Prescription`, `Test`, `Medical_R`, `Prescription_R`, `Financial_R`, and `Patient`. `Visit`, `Prescription`, and `Test` are subtypes of `Item`, for different medical procedures for a patient. The profiles for OT `Medical_R`, attribute `Medical_History`, and method `Read_Med_Rec` are shown in Figure 2. Note that some information in profiles is designer-supplied (e.g., each MP) while for others, input/choices by the designer automatically update relevant profiles (e.g., each MP/AP is included in OTP).

2.2 Relationship-Type Specification

In the second phase of ADAM, the designer can define the relationship types (RTs) between different OTs via a *relationship-type profile (RTP)*. An RTP is a specialized OTP that contains:

1. all information from an OT profile
2. the relationship variant (e.g., one-to-one, one-to-many, set, etc.)
3. the involved source and destination OTs

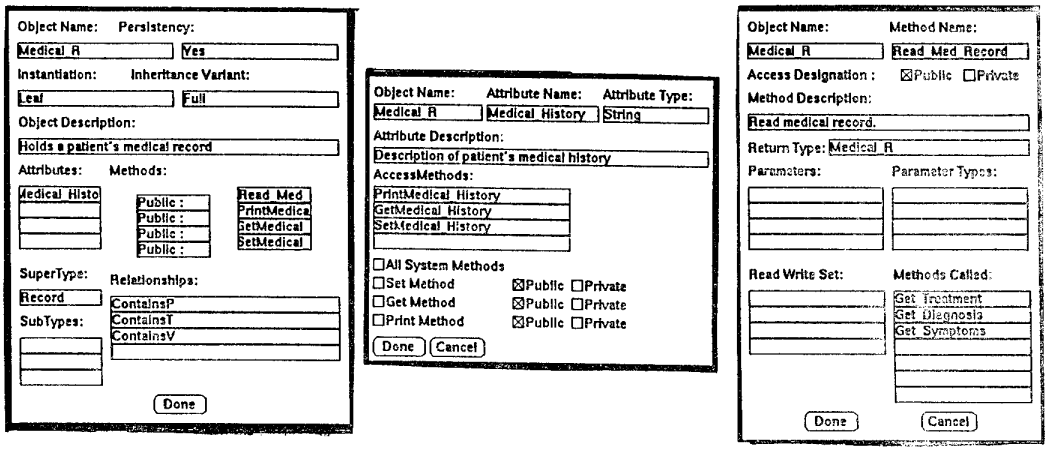


Figure 2: Profiles for `Medical_R`, `Medical_History`, and `Read_Med_Rec`.

Note that except for relationship name and description, all other information in the profile is automatically generated based on a designer's actions. A partial subset of the relationships between the OTs given in Figure 1 is shown in Figure 3. In this figure, we have established associations between the different OTs. For example, a `Medical_R(ecord)` contains the `Visits`, `Prescriptions`, and `Tests` that catalog the complete medical history of the `Patient`.

2.3 URDH Specification

To support URBS, the user-role definition hierarchy (URDH) characterizes the different kinds of individuals (and groups) who all require different levels of access to an application. The responsibilities of individuals are divided into three distinct levels of abstraction for the URDH: user roles, user types, and user classes. *User roles* allow the security software engineer to assign particular privileges to individual roles. To represent common responsibilities among user roles, a *user type* can be defined. Privileges that are assigned to a user type are systematically passed to all of its roles. The different user types of an application can be grouped in to one or more *user classes*. Privileges that are supplied to each class are passed on to its types and their roles.

Figure 4 shows a partial URDH created in the ADAM environment for the HCA, with a more complete URDH given elsewhere [10]. In the figure, the roles are defined in a two-step process of specialization (top-down) and generalization (bottom-up). From a top-down perspective in Figure 4, there are two different user types: `Nurse` and `Physician`. In this case, the software engineer is assuming that each of these user types may have privileges that would be common to all user roles under the type. Within each user type, one or more user roles may be defined. For example, in Figure 4, user roles for `Nurse` include `Staff_RN`, `Discharge_Plng (planning)`, `Education`, and

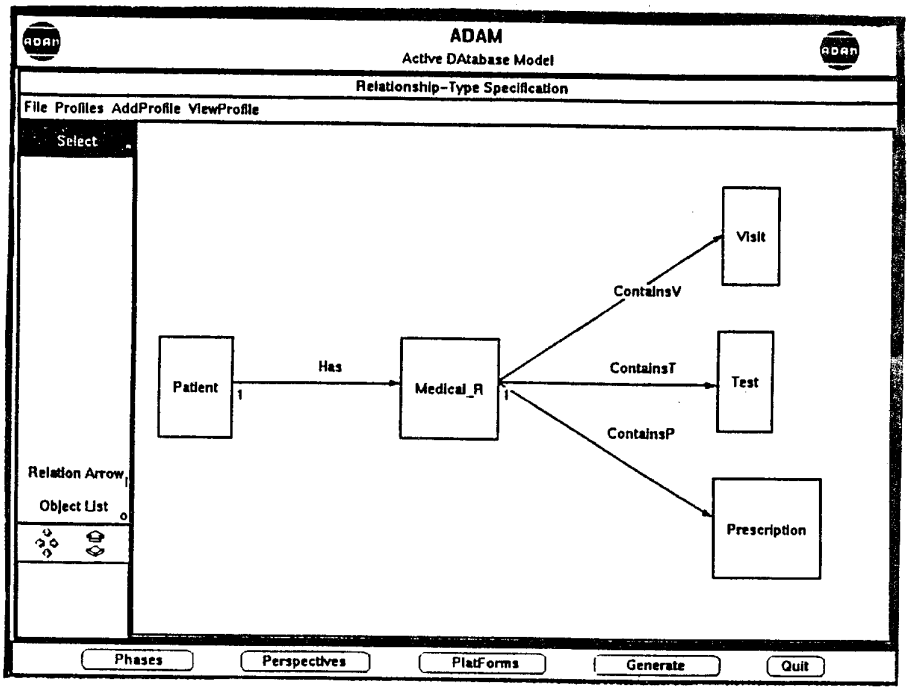


Figure 3: A Subset of the Possible Relationships.

`Manager`. The URDH can also be examined from a bottom-up perspective to determine the common characteristics by the grouping of the user types into user classes such as `Medical_Staff`, `Support_Staff`, and `Other`, which is not shown in this figure.

To more accurately characterize the capabilities of user classes, user types, and user roles in the URDH, with respect to the privileges to be granted against the application, we propose the creation of a *node profile (NP)*. A node profile contains:

1. a name for the node (user role, user type, or user class)
2. a prose description of its responsibility
3. a set of assigned methods (the positive privileges)
4. a set of prohibited methods (the negative privileges)
5. a set of criteria for relating URDH nodes

A *user-class profile or user-type profile* is a specialized node profile. A *user-role profile* is a specialized node profile that also contains a prose description of its security requirements.

In the URDH-specification phase of ADAM, the designer must select the node type (user role, user type, user class) to define a new URDH node as shown in Figure 5. After selecting the type of a node, the designer must supply the node name and node description for the created node. The initial information for the node profile of the user role `Staff_RN` is shown in Figure 6. After a node is created, the designer can select

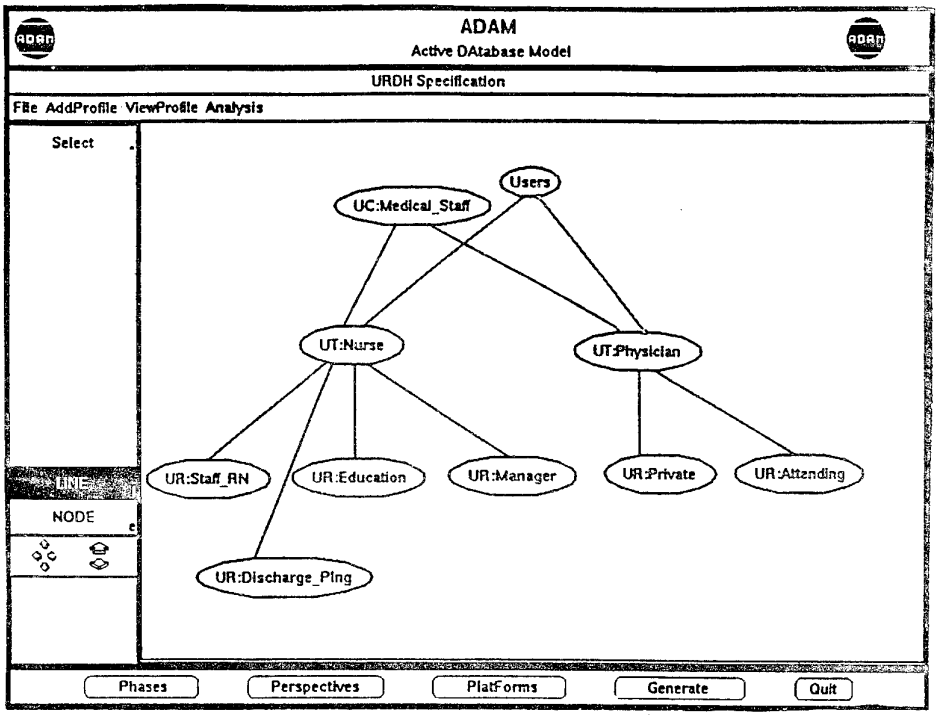


Figure 4: The URDH of the HCA.

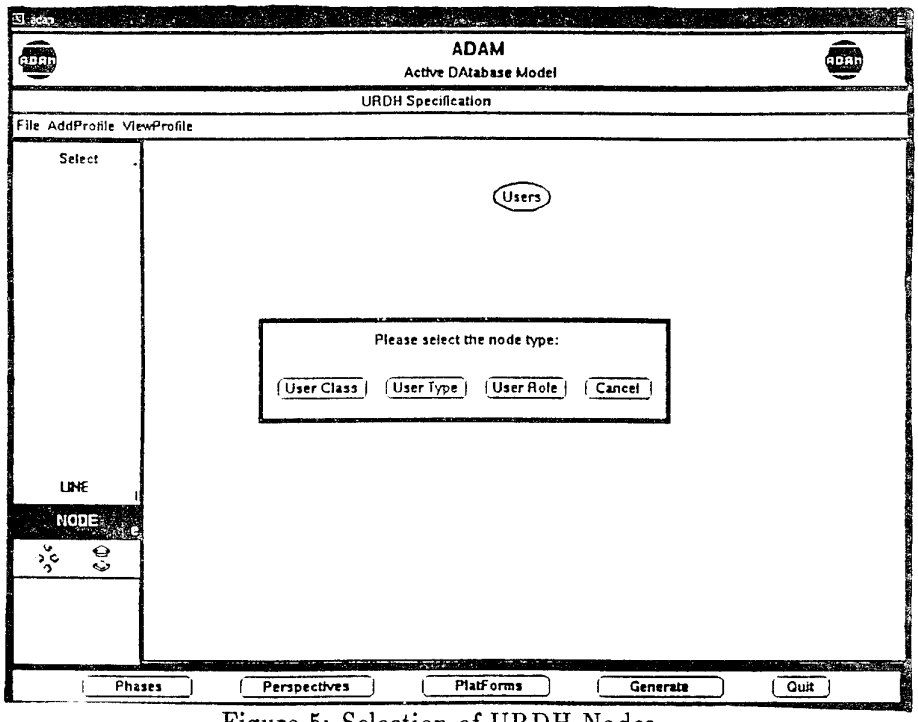


Figure 5: Selection of URDH Nodes.

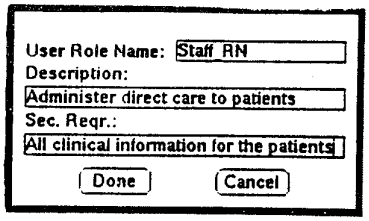


Figure 6: Initial Information for the User Role `Staff_RN`.

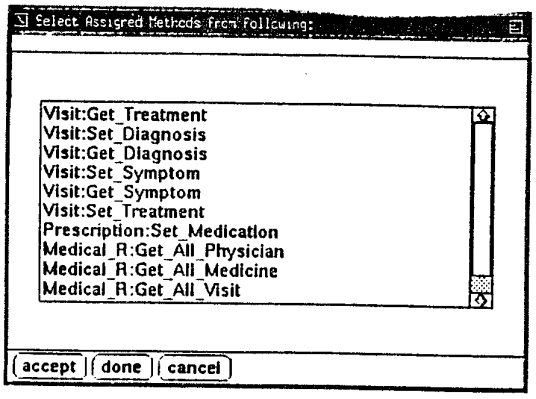


Figure 7: Selection of Assigned Methods for the User Role `Staff_RN`.

the menu option *AddProfile* to supply other information on the security privileges for a node, i.e., assigned and prohibited methods, and consistency criteria. The designer utilizes the mouse to select the assigned/prohibited methods from a list of previously defined methods. To specify the equivalence/subsumption criteria, the designer utilizes the mouse to select nodes from the list of defined URDH nodes. A selection of assigned methods for user role `Staff_RN` is shown in Figure 7.

In the URDH-specification phase of ADAM, the checking on assigned/prohibited methods is performed automatically to insure that there are no conflicts, e.g., an assigned method conflicts with an earlier prohibited method. If a problem is identified by the analyses, the system will not accept the specification and will require a correction by the designer, as shown in Figure 8. Note that the conflict may be more subtle, and arise due to nested method calls, e.g., one assigned method calls a method that calls a method that is prohibited. The checking on consistency criteria is also performed automatically to insure that there are no conflicts when designers specify the assigned/prohibited methods and/or the consistency criteria, as shown in Figure 9.

The complete node profiles for the user role `Staff_RN` and `Manager` are shown in Figure 10. The node description and node security requirements for `Staff_RN` were:

Node Description: *Administer direct care to patients and implement the physician treatment plan.*

Security Requirement: *All clinical information for the patients that they are respon-*

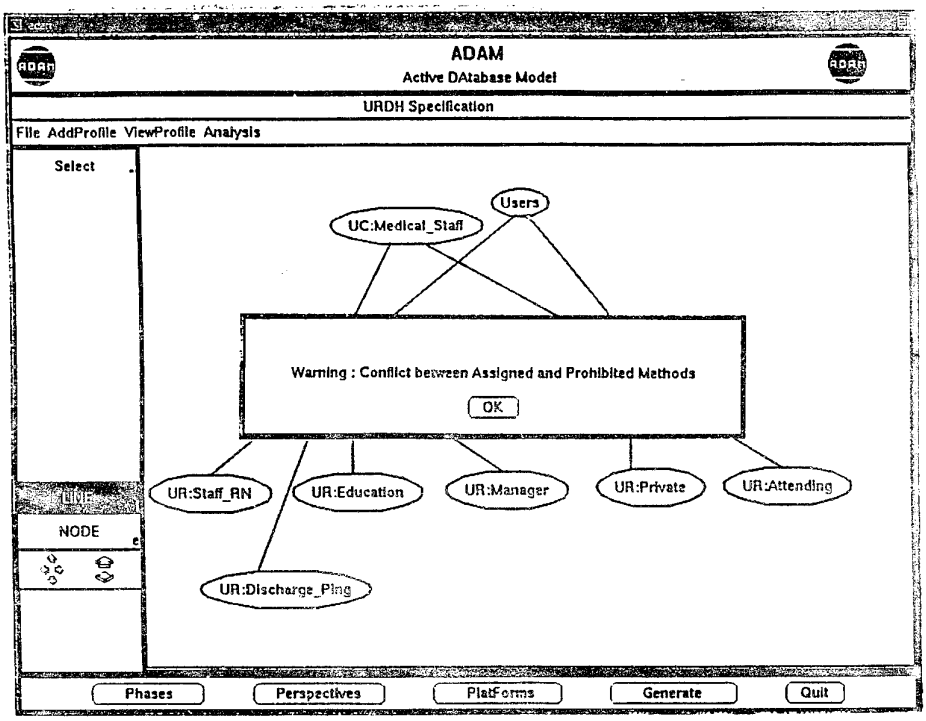


Figure 8: Conflict Identification Message.

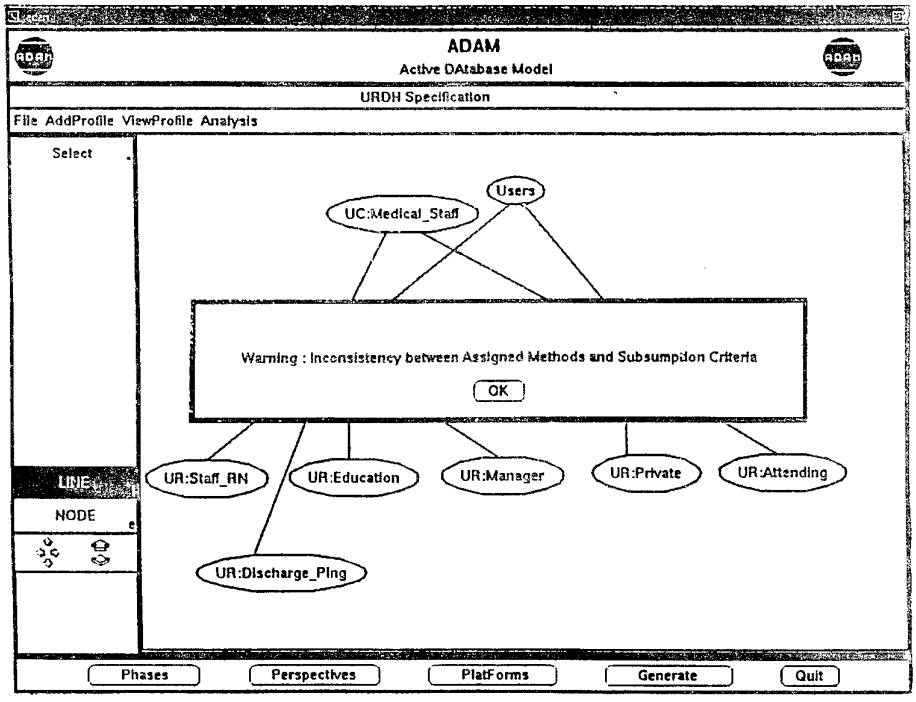


Figure 9: Consistency Criteria Checking Message.

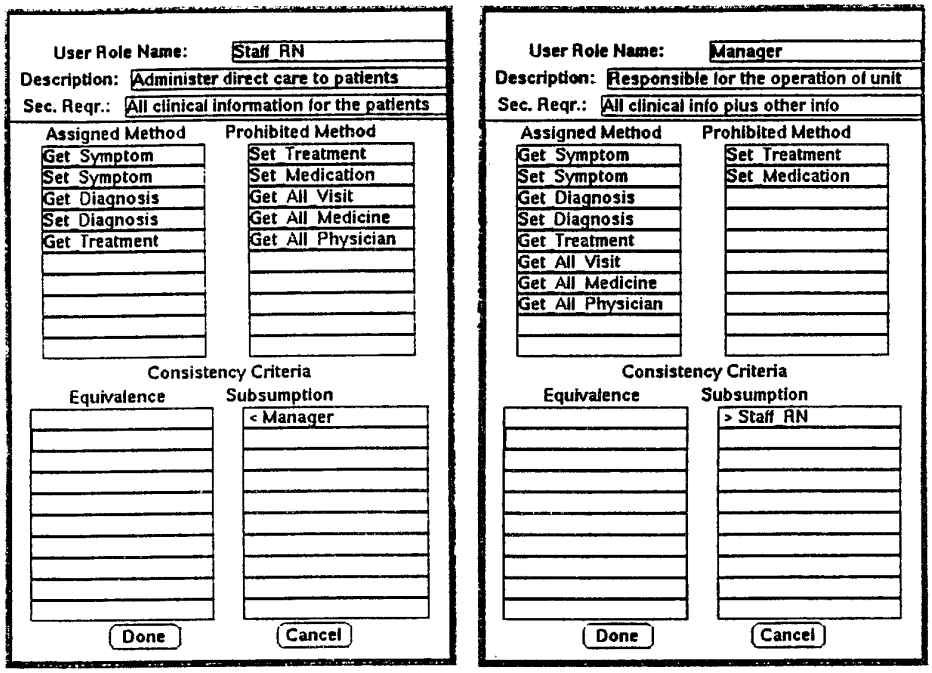


Figure 10: Node Profiles of the User Roles `Staff_RN` and `Manager`.

sible for (referred to subsequently as clinical info.). Can write/modify a substantial portion of clinical information to record the results/patient progress. Cannot change a Physician's orders on a patient.

Other node profiles for the user type `Nurse` and user class `Medical_Staff` are given in Figure 11.

2.4 Authorization-List Specification

To more accurately characterize the capabilities of users in an application, with respect to the privileges to be granted, we employ user profiles, which are similar in concept to node profiles (see Section 2.3 again). A *user profile (UP)* contains:

1. a name for the user
2. a prose description of its responsibility
3. a prose description of its security requirements.
4. a set of assigned roles (the positive privileges)
5. a set of prohibited methods (the negative privileges)
6. a set of criteria for relating users

In the authorization-list-specification phase of ADAM, the designer must supply the user name, user description, and user security requirements when creating a new user.

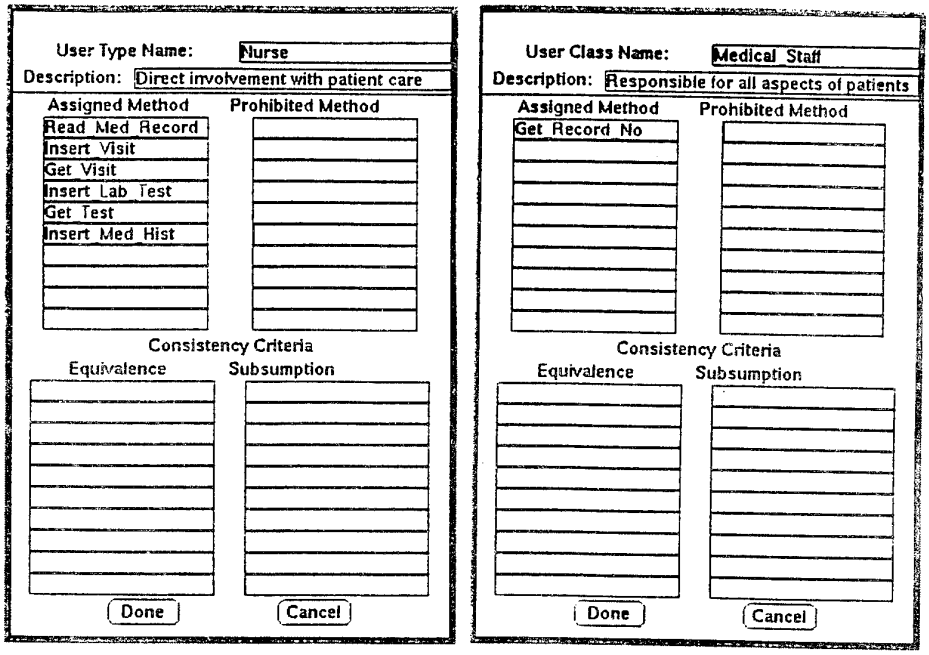


Figure 11: Node Profiles of the User Type `Nurse` and the User Class `Medical_Staff`.

Since all of the actions performed in this phase are very similar to the URDH phase, we omit bit maps. However, note that conceptually, a user has privileges via a set of one or more assigned roles. So, the URDH information is aggregated for each role for which a user has been authorized (or prohibited).

3 Security Analyses

To provide the designer with the ability to compare/contrast the privileges which have been defined (via URDH and/or authorization list) with the application's intended security requirements, an analyses framework is supported. In this section, we briefly review the analyses that have been implemented in ADAM. The research motivation and algorithmic techniques for these analyses were reported in an earlier work [11]. Recall that the analyses are supported in the semantic perspectives of ADAM, as described in Section 2. We provide bit maps of ADAM to demonstrate a select subset of the supported analyses.

3.1 Analyses of User-Role Definition Hierarchy

ADAM supports designer-initiated analyses of the URDH in two categories:

- the capabilities of the URDH node based on the assigned and prohibited methods

- the authorization analysis of the application based on the assigned and prohibited methods

In the first category, the security designer chooses a URDH node (say `Staff_RN`) and can then analyze its privileges with respect to OTs, methods, and private data that can/cannot be accessed. In the second category, an aspect (OT, method, attribute) of the application is selected and the different user roles that have access are supplied.

All of these and later analyses occur at direct and indirect levels. Direct analyses inspect only the explicit privileges that have been established. Indirect analyses search for privileges exhaustively, since methods can call other methods (see the method profile definition in Section 2.1). These nested method calls are important, since they provide an inferred access to methods, OTs, and private data they may not have been intended by the security designer. Indirect analyses are also used to support automatic analyses as discussed in Section 2.3. Due to space limitation, we will omit indirect analyses from our remaining discussion.

3.1.1 Capabilities Analyses

Capabilities analyses allow the security designer to review the permissions given to a chosen URDH node on an application's OTs, methods, and/or private data. This review can occur throughout the time period when the designer is defining the URDH and establishing assigned/prohibited methods for its nodes. For example, the designer can choose the `Staff_RN` node and be presented with:

- all methods which have been assigned to `Staff_RN` and its ancestors (`Nurse`, `Medical_Staff`, and `Users`);
- all OTs which can be accessed by `Staff_RN`, since each assigned method belongs uniquely to a single OT; and
- all private data which is accessed by `Staff_RN`, since each assigned method uses private data in a read, write, or read/write fashion.

Analysis is also available for the prohibited methods to find what cannot be accessed. For example, the designer can choose the `Staff_RN` node and be presented with:

- all methods which have been prohibited to `Staff_RN` and its ancestors (`Nurse`, `Medical_Staff`, and `Users`);
- all OTs which cannot be accessed by `Staff_RN`, since each prohibited method belongs uniquely to a single OT; and
- all private data which is not accessed by `Staff_RN`, since each prohibited method uses private data in a read, write, or read/write fashion.

In the semantic perspective of the URDH, a list of available analyses can be enabled as shown in Figure 12. The designer can select any option from the list and perform the desired analyses. If the "Direct Assigned Methods" option is selected, a set of assigned methods on a selected node will be returned. The results of the direct analysis of

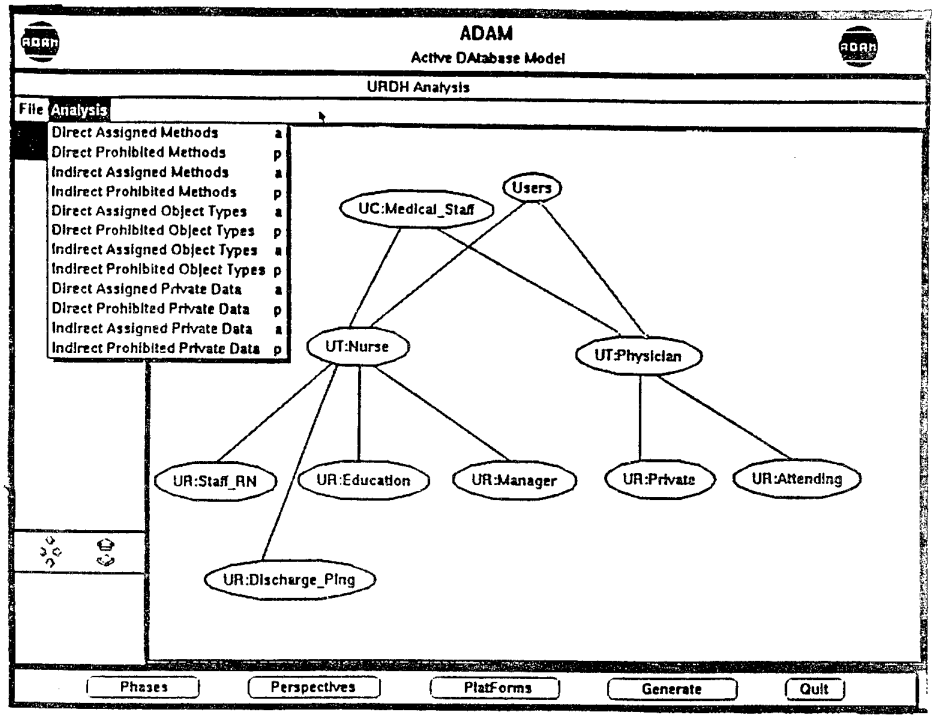


Figure 12: Available Capabilities Analyses of the URDH.

assigned methods for `Staff_RN` is shown in Figure 13. Each method name is associated with the URDH node name, so that the designer can understand where the methods have been assigned. If the method list cannot fit into one window, the designer can use the scroll bar to check other methods. If the designer identifies any problem(s) in method assignments (or the privileges) as a result of the analyses, correction(s) can be made by modifying the URDH, node profiles, or/and the application. Direct analyses for user types work in a similar fashion as shown for `Nurse` in Figure 14. Correspondingly, when the "Direct Prohibited Methods" option is selected, a set of prohibited methods on a selected node will be returned as indicated for `Staff_RN` in Figure 15.

3.1.2 Authorization Analyses

Authorization analyses allow the designer to investigate which user roles have what kinds of access to different aspects of an application (i.e., an OT, a method, or a private data item). For example, the designer can choose the `Medical_R` OT and be presented with:

- all user roles which have access to the `Medical_R` OT;
- all user roles which have access to the methods of `Medical_R` OT; and
- all user roles which have access to the private data items of `Medical_R` OT.

The authorization analyses of prohibited methods are similar to the analyses of assigned methods. For example, the designer can choose the `Medical_R` OT and be presented

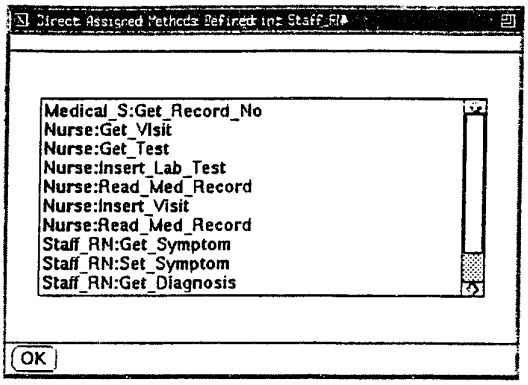


Figure 13: Direct Analysis on Assigned Methods for the User Role `Staff_RN`.

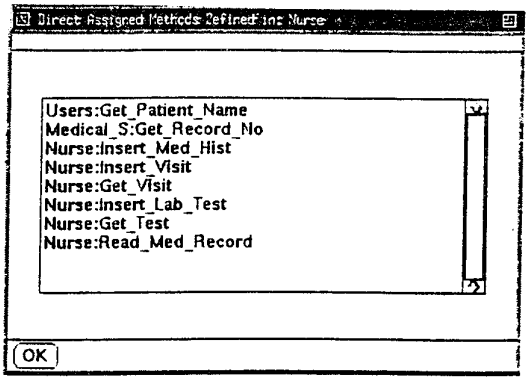


Figure 14: Direct Analysis on Assigned Methods for the User Type `Nurse`.

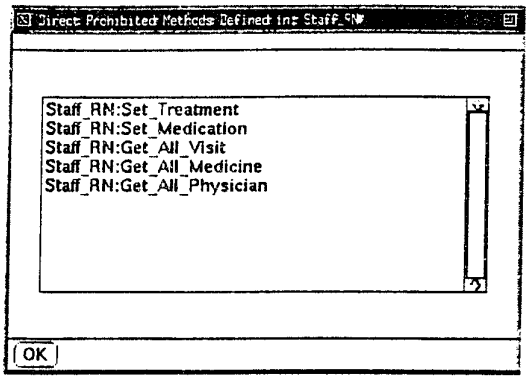


Figure 15: Direct Analysis on Prohibited Methods for the User Role `Staff_RN`.

with a list of user roles that should not be allowed access to the OT. Clearly, these analyses are the complement of capabilities case, and are being implemented.

3.2 Analyses of Authorization List

ADAM also supports a semantic perspective for analyses based on the authorization-list, where multiple (one or more) user roles have been assigned (and/or prohibited) to each individual who accesses an application. These analyses extend the URDH situation, since they (in most cases) repeatedly call the "relevant" URDH analyses for each user role assigned to an individual. The designer-initiated analyses provided for the authorization-list are:

- the capabilities of the individual based on the assigned and prohibited roles
- the authorization analysis of the application based on the assigned and prohibited roles

We do not provide bit maps from ADAM for the authorization list analyses, since these analyses are extensions from the URDH analyses with aggregated information from the user roles.

3.3 Other Analyses

In addition to the security related analyses that have been described, we have also developed a significant set of analysis techniques for non-security object-oriented design model constructs. For example, one analysis would choose OT in an application and perform a "neighborhood search" to all other OTs and RTs that are within a certain distance. This analysis searches for both inheritance and relationship links to determine the correct neighborhood. Another analysis that is supported examines the design of an application to identify cycles that have been caused by different RTs linking OTs. Cycle detection is an important step, especially since an ADAM design might span multiple screens so that an engineer would not "see" all of the interdependencies. These and other analyses have been detailed elsewhere [8].

4 Ongoing Research and Prototyping Issues

This section considers other relevant issues which involve research and prototyping efforts that are in progress and have a strong relationship to security concepts. Specifically, in the first part of this section, we review research on integrity constraints. Integrity constraints represent a research focus [9] that has not been reported to date. Next, we discuss a plausible approach for realizing and enforcing security in a manner that is consistent with object-oriented precepts and principles.

4.1 Integrity Constraints

The integrity constraint (IC) construct for ADAM has been designed to span both programming (typing) and database (derived values) requirements for integrity in an object-oriented model. In our model, we take the view that ICs are restricted to within an OT and define the values that an attribute may take. This fulfills the encapsulation characteristic of the object-oriented paradigm and ensures that all instances of the same OT have identical behavior. An IC applies to a single instance of an OT, i.e., a constraint may not involve the private data items of two separate instances even if the two instances are of the same OT. While this is a very strict definition of dependent behavior, note that the propagation construct in ADAM [6], which has not been discussed herein, is available to handle more complex situations.

Integrity constraints are inherited within the ISA hierarchy, where all constraints defined on the attributes of an OT's ancestors are inherited by the OT itself. In addition, an IC may apply to the private data directly defined on an OT and/or to the private data that the OT has inherited from its ancestor(s). In this sense, an OT is composed of itself and its ancestors; while it appears that a constraint may cross multiple OTs, in reality only one OT (and its instance that contains instances of ancestors) is involved. To maintain consistency with Section 2, an *integrity constraint profile, ICP,* contains:

1. the name of the constraint
2. a prose *IC-description* for the purpose of the IC
3. the constraint variant
4. an algebraic expression describing the constraint
5. the attribute profiles for all attributes involved in the constraint
6. the method profiles for methods impacted by the IC

There are two IC variants: value restriction and attribute dependent. The *value-restriction* variant of ICs restricts the values that an attribute can have based on an empirical value, e.g., `Age` $<$ `25`. In the *attribute-dependent* variant of ICs, the values that the dependent attribute may assume are restricted based on one or more other attributes defined on the same OT (or its ancestors), e.g., `AmtDue = Charge - Insurance`.

An important aspect of ICs that is related to security constraints involves our implementation approach for constraint maintenance. The main thrust of our approach is to ensure the integrity of all attributes on a method-by-method basis. The designer-defined methods specified for an OT are the unit of integrity assurance, and operate by checking the values of all attributes written by the designer-defined method after the execution of every method. Since ICs are limited to a single OT, and may not span instances of OTs not related by inheritance, a more centralized approach to integrity maintenance is not required.

We use a straightforward example from our HCA to demonstrate our approach. Suppose that the following ICs have been defined on the `Prescription` OT for its `Cost` and `W_Sale_Cost` attributes: `IC1: Cost` $>$ `0` and `IC2: W_Sale_Cost` $<$ `Cost`. Assume that the `UpdateCost` method modifies both the `Cost` and `W_Sale_Cost` attributes by the same amount. The first step in constraint maintenance generates a boolean method

for every attribute that is involved in a particular IC. This method is passed the new value for the involved attribute and returns true or false depending on whether the IC has been violated. Using the previous example, a single method is produced for IC1 and two methods are produced for IC2.

```
Boolean CostCheckIC1(NewCst) {Return (NewCst > 0);}
Boolean CostCheckIC2(NewCst) {Return (W_Sale_Cost < NewCst);}
Boolean W_Sale_Cost(NewW_Sale_Cost) {Return (NewW_Sale_Cost < Cost);}
```

The next step is to group together all of the boolean methods that involve the same attribute so that all related ICs can be checked with a single method call, to, in this case, CheckCost. Note that the results of each of the individual boolean methods are logically ANDed together.

```
Boolean CheckCost(NewCst)
        { Return(CostCheckIC1(NewCst) AND CostCheckIC2(NewCst)); }
```

These different methods will be automatically generated by ADAM for maintaining constraints, in conjunction with an appropriate runtime process.

Specifically, we utilize a *constraint consistency OT* (CCOT), which is responsible for maintaining information consistency with respect to an OT's ICs at runtime. Every designer-defined OT contains a CCOT generated by ADAM that operates conceptually in a fashion similar to a constructor (i.e., an instance of CCOT is created whenever an OT is instantiated to track and verify the changes to attributes against defined OTs). The constructor for an OT is extended so that when an instance of an OT is instantiated, a constructor for the CCOT is called to automatically create a CCOT instance. In this manner, an object's constraints are maintained while the vehicle which carries out the maintenance is shielded from the user.

The CCOT is activated on each method call. Any changes to attribute values made to by the method are stored to copies of the attributes located in the constraint consistency object. At the end of the method's execution, the CCOT calls an attribute check method for all attributes modified by the method. If the new values of all attributes (the copies) are consistent with respect to their ICs, the attributes on the original object are updated to reflect the new values, i.e., copy from CCOT to the OT. If any of these checks fail, the effect is that the method did not execute since the values for CCOT are not copied back to the OT.

In our example, if the UpdateCost method is called, the constraint checker first makes copies of the W_Sale_Cost and Cost attribute values. Next, the UpdateCost method is executed, making changes to the copies of the attributes located on the CCOT. When the method has finished executing, the CheckCost and CheckW_Sale_Cost methods are called and their results logically ANDed together. If the results of all of these methods are true, then the values for Cost and W_Sale_Cost are copied to the apropos attribute values located in the original object (instance).

The use of a CCOT encapsulates the integrity maintenance behavior within an OT while providing separation between the behavior and its maintenance mechanism, and therefore offers another layer of information protection. The approach also resolves the

undo problem; since changes are not made to actual attributes until their integrity has been validated, there are no actions to be undone, and the CCOT instance is discarded.

Some may argue that our approach wastes time by delaying the validation of attribute values to the end of a method call. However, this is necessary to avoid "transient inconsistency", where a value is inconsistent for a brief period during a method's execution. For example:

```
1       UpdateCost(amount)
2          {
3            W_Sale_Cost = W_Sale_Cost + amount;
4            Cost = Cost + amount;
5          }
```

suppose that the original values for the attributes are `Cost` = 4 and `W_Sale_Cost` = 3, and the `amount` parameter contains 2. If we were checking attribute integrity on a line-by-line basis, when `W_Sale_Cost` is set to 5 at line 3, the `W_Sale_Cost` < `Cost` integrity constraint would no longer hold. Using our approach, both `W_Sale_Cost` and `Cost` are checked at the end of the method when both will contain consistent values with respect to their ICs.

Note that by making changes to the CCOT and then transferring values to the OT, we are assuming that problems are likely to occur that requiring undoing. This behavior is appropriate for applications where integrity violations are highly probable. An alternative approach would revise the concepts so that the CCOT contains the original values and the OT itself is directly modified. When an undo was necessary, the CCOT values would be copied to the OT. This case applies well to applications where the ICs are not likely to be violated. We believe that both approaches are desirable and will be explored as future research.

4.2 Security Realization and Enforcement

In a recent effort [3], we advocated that the characteristics of the object-oriented paradigm must be the guiding factor in the design and development of security capabilities. This includes: *basic features* such as the public and private interfaces, encapsulation, and hiding; *advanced features* such as polymorphism, dispatching, and overloading; and *paradigm claims* such as software reuse and evolution. In self-critiquing our own efforts, we asked two important questions:

- How can and should these three advanced features be utilized to realize security?
- What role can the paradigm claims play in the security enforcement process?

Polymorphism, through its type independence of code, might be the vehicle by which security code for object-oriented systems can be successfully implemented and reused. In a URBS solution to security, different roles must all undergo the same processes of granting privileges, authentication, and enforcement. When establishing a security policy for an application, polymorphism can be used to develop class libraries for supporting these processes, that are parameterized by type (in this case, user role!). Dispatching

and overloading are strongly linked, and together allow an executing piece of object-oriented code to behave differently based on the type of the invoking instance. There is a strong parallel from a security perspective; dispatching and overloading have strong ties to promoting and supporting the execution of security code via the runtime invocation of different methods based on the involved user role. In this case, the security policy and its associated code can be extended and modified as needed when user roles (or their capabilities) change over time. The common theme of all three advanced features is to consider the design and development of *security class libraries*, which are geared towards the support of security requirements in an object-oriented domain.

The paradigm claims that appear to have the most impact on security for object-oriented systems are software reuse and evolution. In practice, these two claims are tightly linked to the definition and maintenance of OT/class libraries for object-oriented applications. The security solution that utilizes an OT/class library approach is strongly tied to reuse, since once defined, these libraries can be reused as is, extended with new capabilities, or evolved to satisfy changing needs. For a given application (like HCA), apropos security libraries would be automatically included. These libraries would provide all aspects of security, such as definition, authentication, and enforcement. For example, in HCA, a software tool to monitor and establish treatment was to be developed for all professionals that administer care, e.g., nurses, physicians, technicians, etc. In a URBS approach, each of these professionals would have different user roles. The overall security policy for such an application would need to consider and distinguish the security requirements for each role. If such a policy for HCA implemented as a class library, then the user role for physician would be given more expansive access to the library (to allow doctors to set medication and treatment) than nurses. When such a policy is included in the software tool, the end result is that the tool behaves differently based on the user and his/her role (dispatching again).

To realize the aforementioned scenario of a class library for security, where the same tool would operate differently depending on the user role, there must be support at the implementation level in the definition of OTs/classes. The integrity constraint implementation approach (see Section 4.1) can be exploited to further expand the capabilities of the constructor to include instances of the relevant security classes, to define the security policy. This is analogous to the CCOT and provides a way to bridge the gap from type-level security to its instance-level realization. Another choice would be to add a *security constructor* to an OT/class, that would specifically and uniquely embody the security policy. Regardless of the final choice, the idea of a security class library, and its inclusion and reuse both within and across applications, can be strongly advocated as consistent with object-oriented precepts and principles.

5 Concluding Remarks and Future Work

This paper has presented a report on the prototyping of ADAM, a unified environment for supporting object-oriented design and analyses which includes URBS for DAC. While the core concepts and constructs for our object-oriented design model (see Sections 2.1

and 2.2 again) were presented, our major emphases were on the user-role definition hierarchy (for defining roles and establishing privileges in Section 2.3), the authorization list (for individuals who need to play multiple roles in Section 2.4), and the available analyses (see Section 3) within ADAM. These analyses are critical for successful design, since they allow the security designer to compare and contrast the realized design against his/her intended security requirements. This results in designs which are more accurate and precise, at least when considered from a URBS perspective. We also provided a preliminary report on our research/prototyping efforts for integrity constraints (see Section 4.1), and related this work to security realization and enforcement issues (see Section 4.2). Overall, we plan on continuing to develop and evolve ADAM, using it as a test-bed to explore and verify our different research ideas related to both structural and URBS design.

A number of projects related to ADAM have been identified and are ongoing:

- The Impact of Changes Across the Environment: In this case, we are interested in what happens when a significant change to the application occurs. For example, if a user role is deleted, the impact on the authorization list must be considered. Likewise, if methods or object types are deleted, then the URDH and the authorization list may be affected. The issue is the degree to which these changes can be automated within ADAM.
- Enforcement of Roles and Authorizations: Throughout the entire environment, software engineers involved in cooperative design or development on an application should be only able to see, use, and/or modify the methods and object types that have been authorized to them based on their roles. This must be enforced in all relevant portions of ADAM.
- Automatic Documentation Generation: Originally, the different portions of profiles (see Section 2 again) are utilized to create comments in the generated code. This capability hass been extended to automatically create Latex documentation for a particular design.

Our overall goal is to have a unified environment that supports all aspects of software design, development, and implementation, with security as a critical component.

References

[1] H. H. Bruggemann, "Rights in an Object-Oriented Environment", in *Database Security, V: Status and Prospects*, C. Landwehr and S. Jajodia (eds.), North-Holland, 1992.

[2] S. Demurjian, M.-Y. Hu, T.C. Ting, and D. Kleinman, "Towards an Authorization Mechanism for User-Role Based Security in an Object-Oriented Design Model", *Proc. of 1993 Phoenix Conf. on Computers and Communications*, Scottsdale, AZ, March 1993.

[3] S. Demurjian and T.C. Ting, "The Factors that Influence Apropos Security Approaches for the Object-Oriented Paradigm", *Workshops in Computing*, Springer-Verlag, 1994.

[4] K. El Guemhioui, S. Demurjian, and T. Peters, "Object-Oriented Design and Automatic Ada Code Generation in the Education of Software Engineers", *Proc. of 1993 TriAda Conf.*, Seattle, WA, Sept. 1993.

[5] K. El Guemhioui, S. Demurjian, T. Peters, and H. Ellis, "Profiling in an Object-Oriented Design Environment that Supports Ada 9X and Ada 83 Code Generation", *Proc. of 1994 TriAda Conf.*, Baltimore, MD, Sept. 1994.

[6] H. Ellis, S. Demurjian, F. Maryanski, G. Beshers, and J. Peckham, "Extending the Behavioral Capabilities of the Object-Oriented Paradigm with an Active Model of Propagation", *Proc. of the 18th Annual ACM Computer Science Conf.*, Feb. 1990.

[7] H. Ellis and S. Demurjian, "ADAM: A Graphical, Object-Oriented Database Design Tool and Code Generator", *Proc. of the 19th Annual ACM Computer Science Conf.*, March 1991.

[8] H. Ellis and S. Demurjian, "Object-Oriented Design and Analyses for Advanced Application Development - Progress Towards a New Frontier", accepted for publication in *Proc. of the 21st Annual ACM Computer Science Conf.*, Feb. 1993.

[9] H.J.C. Ellis, "An Information Engineering Approach to Unified Object-Oriented Design and Analyses", *Ph.D. Degree Dissertation*, The University of Connecticut, May 1994.

[10] M.-Y. Hu, S. Demurjian, and T.C. Ting, "User-Role Based Security Profiles for an Object-Oriented Design Model", in *Database Security, VI: Status and Prospects*, C. Landwehr and B. Thuraisingham (eds.), North-Holland, 1993.

[11] M.-Y. Hu, "Definition, Analyses, and Enforcement of User-Role Based Security in an Object-Oriented Design Model", *Ph.D. Degree Dissertation*, The University of Connecticut, May 1993.

[12] T. Keefe, et al., "A Multilevel Security Model for Object-Oriented Systems", *Proc. of 11th Natl. Computer Security Conf.*, Oct. 1988.

[13] F. H. Lochovsky and C. C. Woo, "Role-Based Security in Data Base Management Systems", in *Database Security: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.

[14] F. Rabitti, et al., "A Model of Authorization for Next Generation Database Systems", *ACM Trans. on Database Systems*, Vol. 16, No. 1, March 1991.

[15] J. Shilling and P. Sweeney, "Three Steps to Views: Extending the Object-Oriented Paradigm", *Proc. of 1989 OOPSLA Conf.*, Oct. 1989.

[16] D. Spooner, "The Impact of Inheritance on Security in Object-Oriented Database Systems", in *Database Security, II: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1989.

[17] B. Thuraisingham, "Mandatory Security in Object-Oriented Database Systems", *Proc. of 1989 OOPSLA Conf.*, Oct. 1989.

[18] T.C. Ting, "A User-Role Based Data Security Approach", in *Database Security: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.

[19] T.C. Ting, S. Demurjian, and M.-Y. Hu, "Requirements, Capabilities, and Functionalities of User-Role Based Security for an Object-Oriented Design Model", in *Database Security, V: Status and Prospects*, C. Landwehr and S. Jajodia (eds.), North-Holland, 1992.

Database architecture:

Chair: M. Schaefer

Arca Systems, Inc., CA

The SINTRA Data Model: Structure and Operations

Oliver Costich
Center for Secure Information Systems,
George Mason University, Fairfax, Virginia 22030,

Myong H. Kang and Judith N. Froscher
Naval Research Laboratory,
Information Technology Division,
Washington, D.C. 20375

Abstract

Relational database systems are based on a powerful abstraction: the relational data model with the relational algebra and update semantics. If the database design (i. e., the way the data is organized) satisfies criteria provided by this foundation, users have assurance that they can retrieve information in a consistent, predictable way. Multilevel secure database systems must not only provide assurance that information is protected based on its sensitivity, but should be based on a data model as sound and complete as the conventional relational model.

In this paper, we present a data model with a relational algebra and update semantics for a multilevel secure database system whose protection mechanisms are provided by the replicated architecture. The approach is to systematically describe the effects of treating security labels as data and to define explicitly the semantics of these data labels for relational database operations. We also briefly compare the SINTRA data model to earlier ones from the SeaView project and their derivations.

1 Introduction

Like all other database systems, multilevel secure database management systems (DBMS) are based upon a data model. Data models originated as a way of describing the structure of data as used in the actual file systems of database management systems. Over time, they have evolved to modeling data from the point of view of the users and of the applications of the database management system.

We take the current view of data model for a database system, i.e., as a set of concepts that can be used to describe the structure of and the operations on a database [Nav92]. The database structure includes the data types, relationships and constraints on the form, or "template," of the database. The database operations include the ways in which the data may be manipulated via retrievals and updates. The operations ought to provide specifically for insertion, deletion, and modification of the data.

The relational data model [Cod70] is a good example of a data model from this perspective. The structure of the relational data model is well known, i.e., attributes, relational schema, etc. The operations of the relational data model are provided, in part, by the relational algebra [Ull82]. SQL completes the operations portion of the data model by incorporating the power of the relational algebra into a query language which also permits insertion, deletion and modification of data. It is from this point of view that we have approached development of a data model for a multilevel secure relational database based on a replicated architecture.

In the world of multilevel secure relational database systems, early data models, which were for the TCB subset architecture, tended to focus on the definition of the structure rather than the operations [Den87, Lun90]. Later work did consider some operations [JaS91], but still relied on a similar set of underlying constraints to define the structure of the data model. Most of these constraints derive from the work done in the development of the SeaView multilevel secure relational database system [Lun90].

In the course of re-examining data models while developing the prototype for SINTRA (a high assurance multilevel secure relational database based on a replicated architecture), we discovered that some of the constraints of these early data models were not necessary, in our opinion, from database functionality or security points of view, but rather seemed to stem from the way in which the architectures separate data to provide mandatory access control.

We have developed a new data model specifically for the SINTRA prototype. Like the earlier data models developed for the TCB subset architecture, the multilevel data model is specified in terms of the conventional relational data model (but for a different reason). In this paper we will describe the structure of the SINTRA data model and the

constraints on it. In some cases the constraints are the same as those of the TCB subset data models. In these cases we will compare and discuss the potential for enforcing the constraint. In other cases, constraints placed upon the TCB subset data model can be weakened considerably or, in some cases, eliminated entirely, thereby improving the functionality of database systems built upon the corresponding data model. In these cases, we demonstrate the desirability of the improvement by example.

This paper is organized as follows. First, we present our criteria for a multilevel secure relational data model and sketches of both the TCB subset and replicated architectures for multilevel secure databases systems. We then examine the structure of the data model, comparing the constraints imposed for each choice of architecture as described above. Finally, we define the semantics of the insert, update, and delete operations for the SINTRA data model and describe how they are done.

2 Criteria for A Multilevel Secure Relational Data Model

Multilevel secure DBMSs are a relatively new concept, and very few products are in the process of evaluation. Most potential users of multilevel secure DBMS are accustomed to relying on system-high databases. (System-high databases are regular untrusted DBMSs in which only users whose clearances dominates "system high" can access the databases.) In such DBMSs, all data that users can legitimately view in the system is available. The data are not security-labeled but all output from such a system is classified "system high" even though some data in it are in reality lower security level data. The security of such systems is assured by clearing all users to system-high.

When users make the transition to multilevel secure relational database systems, they will bring with them many expectations about what these systems will be able to do based on their experience with system-high DBMSs. In developing data models for multilevel secure relational database systems, these expectations should be accommodated without compromising security to the extent possible. Alternatively, from the user's point of view, a multilevel secure DBMS should allow secure access to all information that users need to see and should provide operational capabilities equivalent to those of a system-high DBMS. This means that (1) users should be able to represent *entities* and *associations* among these entities, and (2) user should be able to store the results of operations in relations (i.e., the relational algebra is closed for these operations). It also implies that multilevel equivalents of relation schemas, relational algebra, and perhaps even SQL should be formulated with as few restrictions as possible.

The data model that is presented here attempts to provide as many capabilities as the current system-high databases have. For example, previous data model investigations [JaS91, Den87] do not address the problem of inserting the result of a join operation into another relation (i.e., INSERT INTO ...) and how the new tuples should be classified. The integrity constraints, relational algebra, and update semantics in this paper are influenced by the need that users have for these database operations.

This work also differs from previous multilevel secure data model investigations [JaS91, Den87] in its treatment of the conventional data model [Ull82] as a special case of a multilevel secure data model. For example, if there is only one security level (i.e., system-high database), then this data model behaves in the same way as the conventional data model if security labels are ignored (i.e., All multilevel relational algebra and update operations behave exactly the same as the conventional relational algebra and update operations).

3 The Replicated and TCB Subset Architectures for Multilevel Secure Databases

This section presents only a brief descriptions of both the replicated (SINTRA) and TCB subset architectures. Detailed description of these are readily available elsewhere and are too lengthy to reproduce here; the purpose of this section therefore is to remind those already familiar with those architectures and direct others to appropriate references. Readers familiar with these architectures and basic data model can skip this section.

3.1 TCB Subset Architecture

There are two variants of the basic TCB subset architecture. TCB subset architectures rely on a trusted computing base (TCB) to separate data at various security levels. This approach requires that multilevel objects, relations, and attributes (sometimes) in the case of multilevel secure relational databases, be decomposed into single-level entities which can be protected by the TCB. The user's view of the multilevel object must be recovered from the single-level entities as needed *via* database views.

In the TCB subset architecture, the DBMS runs under the control of a trusted operating system. Each user of the system has a distinct multilevel view of a relation where the classification of the view is dominated by the user's clearance. The DBMS is untrusted and operates at the user's login level; the instance of the DBMS running

on the user's behalf can retrieve data at that level and below via the trusted operating system.

3.1.1 Vertical TCB Subset Architecture (VTSA)

This is the original form of the TCB subset architecture and was originally produced by the SeaView project [Lun90]. The decomposition-recovery procedures have undergone several iterations. Basically the strategy is to embed multilevel relations into the conventional relational structure by creating additional classification attributes to carry the security label of the "real" attribute values (as do all the architectures we consider in this paper). Multilevel relations are then decomposed vertically (and horizontally) into single-level relations using the "real" key values and the classification attributes. The conventional join operation is the primary means of recovering the multilevel relation from the single-level fragments. Details can be found in [Den87, JaS90].

3.1.2 Horizontal TCB Subset Architecture (HTSA)

This variant of the TCB subset architecture is similar to the previous ones except that the decomposition-recovery scheme replaces the join operation of the recovery process with the union operation. This is accomplished by horizontally decomposing the multilevel relations by security level and entering special markers in place of lower level data. The complete description of this model is presented in its entirety [JaS91].

3.2 Replicated (SINTRA) Architecture

The SINTRA architecture relies on physical separation of data by security level and replication of data across security levels to provide high assurance protection of information [KFC92, Kan94].

The replicated (SINTRA) architecture has a trusted frontend and an untrusted backend database systems for each security level. Each backend DBMS contains information at a given security class together with replicated information from each lower backend database. Hence users have access to a single DBMS containing all and only the information they are cleared to see. For example, secret users have access to the secret backend which contains both secret, confidential, and unclassified data, and confidential users have access to the confidential backend with only confidential and unclassified data. Because data is retrieved from only one backend, queries cannot be used by Trojan horse code in user applications to leak information to malicious processes on a lower security level system.

4 The Data Model – Structure

Multilevel secure DBMSs have been generally defined as follows. The protected objects of the secure system are data items and the subjects of the secure system are operations or sequences of operations on the data items. Each subject or object has a security label from a security lattice and the mandatory access control policy (or security policy) is a variant of the standard Bell-LaPadula policy [BeL76]: Subjects may read objects at or below their own security label but write objects only at their own security level.

In this paper, since we are interested in relational systems, the data items are taken to be values of relational attributes. This is commonly called element level labeling (as opposed to tuple level labeling, which is not addressed here). In the remainder of this section, we discuss the structure and constraints of the SINTRA data model and compare/contrast it with the TCB subset architecture data models, both VTSA and HTSA.

4.1 Relational Schema

Given a conventional relational schema

$R(A_1, A_2, \ldots, A_n)$

the corresponding SINTRA multilevel relation scheme is denoted by

$R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TL)$

where each A_i is a data attribute over domain D_i, each C_i is a classification attribute for A_i and TL is the tuple-level attribute. $R(A_1, A_2, \ldots, A_n)$ is the underlying relation as viewed by the user.

Notice that the security label of a given attribute value is the value of another related attribute. Thus the security labels are stored as data in the conventional sense. There is nothing in data model theory that require that the security labels of data items be themselves relational data, but only that a security label be associated. The reason that both SINTRA and the TCB subset architectures take this approach is that both anticipated exploiting conventional relational DBMSs to implement their systems by embedding them in the conventional relational model. In the SINTRA case, the conventional systems are used, unaltered internally, as the backends, while in the TCB subset case, the conventional systems store relational data in files whose separation is assured by the TCB.

Given a tuple t in relation R, $t[X_i]$ denotes the value of attribute X_i in relation R.

We use R ambiguously for both $R(A_1, A_2, \ldots, A_n)$ and $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TL)$ with context being the arbiter.

In the SINTRA schema, the TL attribute represents the security level at which the tuple originated whereas in the TCB subset model, the tuple class is simply the maximum security level of all attributes. Notice that $t[TL]$ in the SINTRA data model is not just the least upper bound of all $t[C_i]$ because complex operations performed at a high security level may generate tuples with all attribute value labels lower than the $t[TL]$. How this can occur will be discussed in section 5.2.1., which describes the operations of the SINTRA model.

4.2 Entity Integrity Constraint

Let A_{PK} be the primary key of relation $R(A_1, A_2, \ldots, A_n)$. A multilevel relation R satisfies entity integrity if and only if for all tuples t in relation R

- $A_i \in A_{PK} \Rightarrow t[A_i] \neq \text{null} \wedge t[C_i] \neq \text{null}$, and
- $t[TL] \geq t[C_i]$ for any i.

Condition (1) is similar to the definition of entity integrity for the untrusted relational databases. Condition (2) requires that if the tuple is generated or modified by a $t[TL]$-user then any element classification within the same tuple should be equal to or lower than $t[TL]$.

The TCB subset models have a more restrictive constraint. In particular they require that the primary key value be uniformly classified. That is,

$A_i, A_j \in A_{PK} \rightarrow t[C_i] = t[C_j]$.

This restriction does reduce functionality. Consider this highly simplified example of a relational system:

EMP(ss#, name, address)
MISSION(mission#, description, skills_needed)
EMP-MISSION(ss#, mission#, location)

The EMP-MISSION relation represents a *many-to-many* association. It is possible that ss# has a value which is classified at a low level while mission# and location are classified higher. But the primary key of the EMP-MISSION relation is {ss#, mission#}, and the

TCB subset data model would prohibit this relation. because the TCB subset models enforce this restriction. In the SINTRA architecture, this restriction is unnecessary. This is important because relations represent both entities and associations, and associations can be more sensitive than the entities whose relationship is defined.

4.3 Null Integrity Constraint

We mention this only for completeness as it is unnecessary for the SINTRA data model. Some versions of the TCB subset model enforce constraints on `nulls`; namely that `nulls` are classified at the level of the key (which is uniformly classified).

In the SINTRA model, classification of `nulls` is determined as for other data by the operations used to enter or modify data.

4.4 Polyinstantiation Integrity Constraint (PI)

For the SINTRA model, let A_{PK} be the primary key of $R(A_1, A_2, \ldots, A_n)$ and let C_{PK} be the corresponding classification attributes in $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TL)$. A multilevel relation R satisfies the polyinstantiation integrity property if and only if for every i, $1 \leq i \leq n$, the functional dependency

$$A_{PK}, C_{PK}, C_i, TL \rightarrow A_i$$

holds. The user specified primary key A_{PK} in conjunction with the classification attributes C_{PK}, C_i and TL uniquely determine the values of attribute A_i. This constraint limits polyinstantiation within a single security class. That is, given values for the primary key and for all the classification attributes, the tuple is determined uniquely.

In the SINTRA model, the polyinstantiation constraint is imposed as an extension of entity integrity, treating $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TL)$ as a conventional relation whose primary key is $A_{PK}, C_1, \ldots, C_n, TL$.

PI is easiest to enforce in the VTSA model because only a single instance of a pair of values $t[A_i, C_i]$ is kept for attribute A_i not in A_{PK}. HTSA requires using a decomposition-recovery technique which is subject to ambiguous interpretation[1]. The SINTRA model has slightly more difficulty enforcing the constraint. Consider the standard example [JaS91] of the relation.

SOD(ship, objective, destination)

[1]The ?-replacement rule of the recovery algorithm does not specify which "replacement" to choose if several are available. Unless this is very carefully done, the original relation is not recovered.

and the following sequence of actions

Operation level		Ship		Objective		Destination		TL	
1. Insert	U	Ent	U	Explore	U	Tolas	U	U	*deleted after action 3*
2. Update	C	Ent	U	Mine	C	Tolas	U	C	
3. Update	U	Ent	U	Explore	U	Sirus	U	U	
4. Update	S	Ent	U	Spy	S	Tolas	U	S	
		Ent	U	Spy	S	Sirus	U	S	

where
- action 2 polyinstantiates the original tuple
- action 3 updates (and replaces) the original tuple
- action 4 updates the existing tuples by entering a secret objective.

The consequence of this sequence of actions produces a pair of S-tuples which do not satisfy the PI constraint.

The resolution of this difficulty in the SINTRA model can be handled by requiring that the update of the `Destination` attribute by action 3 be propagated to the C-level tuples as well, so that the U-labeled `Destination` is always `Sirus`. Thus the S-tuple whose `Destination` is `Tolas` would not appear. In general, an attribute value update must be propagated to all higher level tuples which were derived from the updated tuple by polyinstantiation.

5 The Data Model – Operations

The relational algebra and modification actions for conventional databases are well established, but for multilevel secure databases these operations are not well defined. Indeed, different data models may have different algebras. In this section, we define multilevel relational algebra and modification operations for the SINTRA data model in terms of the relational algebra for conventional databases consistent with our embedding the multilevel relations in the conventional relational structure.

Before doing this we establish some notation. Given a security level c, the c-user's view of a multilevel secure database consists of c-level information and other information from levels strictly dominated by c. Consider, for the time being, a DBMS system with two security classes, high (H) and low (L), where H dominates L. Let *L-user* denote a user session level is L. Let R_c denote the portion of a relation R that is generated by c-users for c = L or H. The relation R in the low untrusted backend database will have only R_L, and all tuples in R_L have tuple-level L. However, the relation R in the high

untrusted backend database contains $R_L \cup R_H$.

For the TCB subset architectures, definitions of relational operations have not been completed. For VTSA, the SeaView final technical report [Sho89] specifies a multilevel SQL (MSQL) subsystem and gives the syntax of `select, update, insert` and `delete` statements but does not specify the actual effect of these on the underlying multilevel relations. For HTSA, `update, insert` and `delete` are defined in [JaS91]. We will specify the equivalents of these and extend them to more complex operations.

5.1 A Multilevel Secure Relational Algebra

As in the conventional relational algebra, there are relations and operations in the multilevel relational algebra. The relations represent both entities and associations. Operations take as arguments one or two relations and produce another relation. We define a very simple relational algebra which is sufficient to give the general idea of how it can be extended. A more extensive relational algebra and multilevel SQL for the SINTRA model are the subject of current research.

We can express the multilevel relational algebra between two multilevel relations R and S in terms of the conventional (single-level) relational algebra among R_L, R_H, S_L and S_H. Relational algebra among R_L, S_L, R_H, and S_H is exactly the same as conventional relational algebra [Ull82] unless otherwise stated. We use the same operator notation for both multilevel relational algebra and the conventional relational algebra, since it is clear from the context which is intended.

Multilevel relational algebra between two multilevel relations R and S in the high backend can be defined as:

- Select (σ) and project (Π) operations act exactly the same as those in conventional relational algebra
- $R \cup S = (R_L \cup R_H) \cup (S_L \cup S_H)$
- $R - S = (R_L - S_L) \cup (R_H - S_H)$
- $R \times S = (R_H \times S_H) \cup (R_H \times S_L) \cup (R_L \times S_H) \cup (R_L \times S_L)$

where the operators on the left side of the definitions are what are being defined and those on the right hand side are those of the conventional relational algebra.

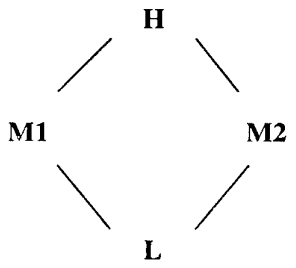
The select and project operations may include the security label attributes in their parameter sets. The cross product operation is particularly significant because it is essential in retrieving related data from multiple relations. The join operations, including

equijoin, outer join, etc., depend directly upon the cross product. These join operations can be defined using the cross product, select, and project operations as usual. It is important to note that without the cross product, the operators which one can use would be limited to operations on single relations, rendering the relational system no more powerful than a flat file database.

Given a relational algebra, an SQL-like retrieval language can be defined. It is also evident that this could be readily extended to an arbitrary security lattice.

However, we believe this relational algebra is overly simplistic as it places the responsibility for the manipulation of data with respect to security labels completely on the user. It is our intention to extend the definition of the relational algebra primitives so that much of the security label specification in queries can be eliminated. In particular, variants of the usual database operations can be defined depending upon whose view of the data a user wishes to see. We will not pursue this in detail in this paper, giving only the following example of two variants of the cross product operator for a specific security lattice.

Consider the following security lattice.



H > M1 > L
H > M2 > L

One cross product (**complete cross product**) at the high (H) security class can be expressed as:

$$
\begin{aligned}
R \times S = \; & \bigcup \{ R_a \times S_b \mid a,b \in \{L, M1, M2, H\} \} = \\
& (R_H \times S_H) \cup (R_H \times S_{M1}) \cup (R_H \times S_{M2}) \cup (R_H \times S_L) \cup \\
& (R_{M1} \times S_H) \cup (R_{M1} \times S_{M1}) \cup (R_{M1} \times S_{M2}) \cup (R_{M1} \times S_L) \cup \\
& (R_{M2} \times S_H) \cup (R_{M2} \times S_{M1}) \cup (R_{M2} \times S_{M2}) \cup (R_{M2} \times S_L) \cup \\
& (R_L \times S_H) \cup (R_L \times S_{M1}) \cup (R_L \times S_{M2}) \cup (R_L \times S_L)
\end{aligned}
$$

Each tuple will have $t[TL] = H$ because an H-level user performs the cross product. For

this operation, the result is the product of all data that the H-level user is entitled to see.

Another cross product (**dominance cross product**) can be defined in the following way:

$$R \times S = \bigcup \{ R_a \times S_b \mid a \geq b \text{ or } a \leq b, \text{ and } a,b \in \{L, M1, M2, H\} \}$$

The dominance cross product allows the H-user to see the data as it would be seen by M1-users, M2-users, and L-users, but nothing more. Thus ($R_{M1} \times S_{M2}$) and ($R_{M2} \times S_{M1}$) are not included in this case because there is no dominance relationship between the two security classes, M1 and M2, and so neither would be visible at either level M1 or M2.

5.2 Data Modification Operations

In this section, we will present the SINTRA equivalents of the insert, update, and delete operations. We use `replace` rather than "update" to avoid confusion with the update operations which are used to maintain the consistency of replicated data. Even though we will use SQL-like syntax to describe the form of these operations, we will use `retrieve` rather than "select" for a similar reason. The `insert` and `replace` operations are defined in both simple and complex forms.

A c-user's view of multilevel secure database consists of c-level information and other information from levels strictly dominated by c. Therefore, we concentrate our discussion on a single user level (e.g., c-level) .

5.2.1 Insert

The insert operation allows users to add new tuples to an existing relation. A c-user cannot insert values in any classification attributes C_i or tuple-level TL. These are all implicitly given the value c, which is the user's login level, by the system.

There are two types of insert operations: (1) simple insert and (2) complex insert. We discuss the interpretation of each operation in the following.

Simple Insert

The simple insert query executed by a c-user, where the access class c is implicitly determined by the user's login class, has the following general form:

```
insert into R [(A1 [, A2, ...])]
values (a1 [, a2, ...])
```

In this notation, the brackets denote optional items and the "..." signifies repetition. If the list of attributes is omitted, it is assumed that all the data attributes in relation R are specified.

Let t be the tuple to be inserted such that if A_i is in the attribute list of `insert` queries, then $t[A_i] = a_i$ and $t[C_i] = c$; otherwise $t[A_i] = null$ and $t[C_i] = c$. The insertion is permitted if and only if:

- $t[A_{PK}]$ contains no nulls (as is necessary to enforce entity integrity constraint).
- For all u in relation R, $u[A_{PK}, C_1, \ldots, C_n, TL] \neq t[A_{PK}, C_1, \ldots, C_n, TL]$.

Otherwise it is rejected, because it would violate polyinstantiation integrity. In other words, a c-user can insert a tuple t in relation R if R does not already have a tuple with the same primary key, attribute classifications, and tuple level classification. Each element-class and tuple-level of a new tuple are set to c.

Complex Insert

The complex insert queries executed by a c-user have the following general form:

```
insert into R [(A1 [, A2, ...])]
retrieve σ1 [, σ2, ...]
from T1 [, T2, ...]
[where cond]
```

where T_i denotes relation name, σ_k signifies any attribute of a relation in the `from` clause or expression, and *cond* may contain any relational algebraic Boolean expression using attributes of tables (including security level attributes) in a `from` clause.

The insertion is permitted and the tuple-level classification of tuples to be inserted will be set to c if and only if:

- For all u in relation R, $u[A_{PK}, C_1, \ldots, C_n, TL] \neq t[A_{PK}, C_1, \ldots, C_n, TL]$, where t is a new tuple to be inserted (i.e., polyinstantiation integrity is preserved).

For example, if the security lattice is {H, L} and an H-user wants a cross product operation between two relations T and S, to be inserted into relation R, then (T $\times$ S) = ($T_H \times S_H$) $\cup$ ($T_H \times S_L$) $\cup$ ($T_L \times S_H$) $\cup$ ($T_L \times S_L$) will be inserted into relation R provided that newly generated tuples do not violate the above condition. The tuple level classification of newly inserted tuples will be *H* because a *H*-user creates those tuples.

Notice that this can create tuples t where the tuple level strictly dominates the security label of every attribute. In particular, every tuple in $T_L \times S_L$ will have the security label of each attribute set to L but the tuple levels will be H. This represents a real world situation where the classification of individual information may be lower than that of the association of the information. An example of this case is give in [Lun89], where EMPLOYEE(EMP#, NAME, DEPT, ...) and PROJECT(PROJ#, BUDGET, ...) are confidential relations while WORK-ON(EMP#, PROJ#) may be a secret relation.

5.2.2 Replace

The replace operation allows users to change data attribute values in existing tuples, but the replicated architecture system does not allow a user to modify values of the classification attributes C_i or the tuple-level attribute *TL*. These are controlled by the mechanisms of the replace operation.

The `replace` queries executed by a c-user have the following two general forms:

Simple Replace

```
replace R
set A_i = s_i [, A_j = s_j, ...]
[where cond ]
```

Complex Replace

```
replace R
set A_i = query_i [, A_j = query_j, ...]
[where cond ]
```

where s_i is an expression, $query_i$ is a `retrieve` query that must return exactly one value in the domain of A_i, and *cond* is a Boolean expression which identifies those tuples in R that are to be modified.

Let $R'_c = \bigcup \{ R_{c'} \mid c' \text{ strictly dominated by } c \}$ and define two sets, S and S':

S = {t $\in$ R_c | t satisfies the *cond* in **where** clause }

S′ = {t $\in$ R'_c | t satisfies the *cond* in **where** clause }

We first consider the case where R = $R_c \cup R'_c$. If $t \in S$ then t will be replaced by the new tuple t' where

$$t'[A_i, C_i] = \begin{cases} < s_i \text{ or } query_i,\ c > & A_i \text{ is in the } \texttt{set} \text{ clause} \\ t[A_i,\ C_i] & otherwise \end{cases}$$

This updates the tuples at the user security level. If t $\in$ S′ then consider the new tuple t″ where

$$t''[A_i, C_i] = \begin{cases} < s_i \text{ or } query_i,\ c > & A_i \text{ is in the } \texttt{set} \text{ clause} \\ t[A_i,\ C_i] & otherwise \end{cases}$$

and $t''[TL] = c$. This determines the new (polyinstantiated) tuples to be produced by the **replace**. If there exists $u \in R_c$ for which $u[A_{PK}, C_1, \ldots, C_n, TL] = t''[A_{PK}, C_1, \ldots, C_n, TL]$, replace u by t''. Else add t'' to R_c where $t''[TL] = c$.

The effect of this rule is to update tuples which may have been derived by polyinstantiation from lower tuple. If the lower level has not already been polyinstantiated it is done at this point.

For instance, consider an H-user's query to do a replace on relation R and the **where** clause contains a join operation between two relations R and T (i.e., R $\bowtie$ T) where R = ($R_L \cup R_H$) and T = ($T_L \cup T_H$). Let two sets, S and S′ be:

S = { t $\in$ Π_{R_H} (σ_{cond} ($R_H \times T$)) }

S′ = { t $\in$ Π_{R_L} (σ_{cond} ($R_L \times T$)) }

The values of tuples in set S will be modified. The tuples in set S′ represent polyinstantiated tuples that will be potentially inserted into R_H or replaced in R_H after appropriate values are modified. In particular, the tuple class will become c.

Now we consider the effect of **replace** on tuples above the user class. Let R″ = $\bigcup$ { $R_{c''}$ | $c'' > c$ }, $t \in$ S, and $A_i \in$ **set** condition. If there is a tuple $t'' \in R''$ that is a polyinstantiation of t, i.e., $t[A_{PK}, C_{PK}] = t''[A_{PK}, C_{PK}]$, and $t''[C_i] = c$ then its value will be modified according to s_i or $query_i$.

5.2.3 Delete

The `delete` queries executed by a c-user have the following general form:

delete from R
[where *cond*]

Tuples that satisfy *cond* in the **where** clause will be deleted from R_c. Said differently, in view of $\star$-property [BeL76], only those tuples t that satisfy *cond* and $t[TL] = c$ are deleted from relation R.

Consider an H-user's query to delete tuples from relation R that satisfy a join condition between two relations R and T (i.e., R $\bowtie$ T) where R = ($R_L \cup R_H$) and T = ($T_L \cup T_H$). The tuples in set S where S = { t $\in \Pi_{R_H}$ (σ_{cond} ($R_H \times$ T)) } will be deleted from R_H.

Even though our examples in this section use relations that have two security levels, we can easily generalize the concept to relations that have a general security structure as we did in section 5.1.

Adding these update operations to the SQL-like retrieval language derived from the relational algebra yields an SQL-like language, capable of performing all basic multilevel relational database operations.

6 Summary

We presented a data model for the SINTRA architecture database system. Based on this data model, a simple multilevel relational algebra and the semantics of update operations for multilevel relation were described.

We believe that any attempt to define multilevel SQL should be based on a multilevel relational algebra and the semantics of multilevel update operations. We plan to investigate multilevel SQL issues in the near future.

References

[BeL76] Bell, D. E., and LaPadula, L. J. Secure computer systems: Unified exposition and multics interpretation. The Mitre Corp. (1976).

[Cod70] Codd, E. F. A relational model for large shared data banks. Communications of the ACM, 13, 6 (1970).

[Den87] Denning, D. E., et el. A multilevel relational data model. IEEE Symposium on Research in Security and Privacy (1987).

[JaS90] Jajodia, S. and Sandhu, R. Polyinstantiation integrity in multilevel databases. IEEE Symposium on Research in Security and Privacy (1990).

[JaS91] Jajodia, S. and Sandhu, R. Toward a multilevel secure relational data model. Proceedings of ACM SIGMOD International Conference on Management of Data (1991).

[KFC92] Kang, M. H., Froscher, J. N., and Costich, O. A practical transaction model and untrusted transaction manager for multilevel-secure database systems. Proceedings of the IFIP 6th Working Conference on Database Security (1992).

[Kan94] Kang, M., et el. Achieving database security through data replication: The SINTRA prototype. Submitted for publication (1994).

[Lun89] Lunt, T., et el. Aggregation and inference: Facts and Fallacies. IEEE Symposium on Research in Security and Privacy (1989).

[Lun90] Lunt, T., et el. The SeaView security model. IEEE Transaction on Software Engineering, 16, 6 (1990).

[Nav92] Navathe, S. B. Evolution of data modeling for databases. Communications of the ACM, 35, 9 (1992).

[Sho89] Shockley, W. R., et el. Secure distributed data views: System specification. RADC-TR-89-313, Vol V (Rome Air Development Center).

[Ull82] Ullman, J. D. Principles of database systems. Computer Science Press (1982).

The b^2/c^3 problem: How big buffers overcome covert channel cynicism in trusted database systems

J. McDermott
Naval Research Laboratory, Code 5542, Washington, DC 23075, USA

Abstract

We present a mechanism for communication from low to high security classes that allows partial acknowledgments and flow control without introducing covert channels. By restricting our mechanism to the problem of maintaining mutual consistency in the replicated architecture database systems, we overcome the negative general results in this problem area. A queueing theory model shows that big buffers can be practical mechanisms for real database systems.

Introduction

Kang and Moskowitz [8] presented a general mechanism for rapid and reliable communication from low to high security classes. The mechanism, called the *Pump*, includes an adjustable and easily quantifiable covert channel to provide acknowledgments. Their general result is that reliability, performance, and security cannot be achieved together. This negative result agrees with other work [10] and we do not dispute it here. Instead, we present positive results for a useful special case of communication from low to high classes: maintenance of mutual consistency in replicated architecture multilevel-secure database systems.

The replicated architecture [6] is an approach to providing strong multilevel security in database systems. It provides multilevel security by replicating single-level copies of low sensitivity data into higher classes. The replicated architecture depends upon the ability to write-up reliably without creating an undesirable information flow. According to the Bell-LaPadula model [1], write-up without read access is permissible. This kind of write-up is performed to volatile storage, without acknowledgment. Furthermore, it requires the use of memory descriptors and mechanisms that do not carry read access permission to the destination memory segment, a feature rarely supported by existing hardware. The latter problem can be overcome by simulating the write-up with a read-down, but the lack of coordination and the volatile nature of the destination memory segment remain problematic.

By exploiting the structure of a computation, Sandhu, Thomas, and Jajodia [13, 14] have shown how write-up without acknowledgment can be used in object-oriented systems. Kang and Moskowitz have proposed a general mechanism for writing up reliably with recovery by using acknowledgment with a controlled bandwidth[1] covert

1. Moskowitz and Kang [12] argue that the concept of bandwidth is not a sufficiently precise measure of the vulnerability introduced by a covert channel and provide a new metric, the *small message criterion (SMC)*. The small message criterion depends on a triple (n, τ, ρ): when a covert channel exists in a system, the SMC gives guidance for what will be tolerated in terms of covertly leaking a short message (e.g. master key) of length n bits in time τ with fidelity of transmission $\rho\%$.

channel. Kang and Moskowitz assert that, for the general case, one cannot have write-up that is reliable, recoverable and secure. The thesis here is that, for an important special case, this is not so. Our special case is write-up performed for the purpose of maintaining mutual consistency in the replicated architecture.

Three advantages of restricting our solution to the replicated architecture are: 1) bounded storage space requirements at the destination, 2) a relatively small number of source and destination processes, 3) transaction management. Because we are only writing up for the purpose of replicating data items in a database, we know that no new objects are created by writing up to higher classes[1]. We can fix the total storage available at lower security classes and thus bound the total replicated storage for all higher security classes. Because we are only supporting database system instances, we know there will not be a large number of readers and writers[2]. Because we are only supporting systems with transaction management capability, we can choose to discard some write-ups in a correct fashion, in the event of a failure, and bring the replicas into convergence with later transactions. This latter point is proved by Bernstein, Hadzilacos, and Goodman [2].

Our specific problem is to provide a service for propagating update projections in the replicated architecture database system. This service is to be *reliable, recoverable,* and *secure*. By secure we mean free from implementation invariant covert channels and compliant with a Bell-LaPadula access control policy, as discussed by [6]. By recoverable we mean that write-ups accepted by the service are completed in the event of a system failure. By reliable we mean that, if a write-up is requested, the requestor can know if the write succeeded or failed, that is, acknowledgments are given to the writer.

We conclude this section with some definitions. In the following sections we review the Pump mechanism, define the basic write-up service, discuss necessary buffer size, present some usability enhancements, and discuss our conclusions.

In our discussion, we assume that all processes use *stable storage* in a recoverable way. Stable storage [2] is storage that is not affected by a system failure, e.g. disk storage. *Volatile storage* is storage that is affected by system failure; system failures cause the loss of possibly all of the contents of volatile storage. Stable and volatile are relative terms; we could consider off-line tape storage as stable and disk storage as volatile because disk hardware failures do not affect the off-line tapes. A more precise definition would distract us from our point. When we say that processes use stable storage in a recoverable way, we mean that they keep their data on stable storage, and follow the usual approaches to logging and caching in volatile storage [2] to ensure that their data can be recovered after a crash.

The Pump

The Pump provides communication from a low source process to a high destination process. It is a trusted mechanism with three components: trusted low buffer *TLB*, trusted high buffer *THB*, and communication buffer *CB*. A low source process sends

1. Yes, there is a problem with multilevel transactions that will be discussed in the conclusion.
2. Readers and writers being database system server/data manager instances.

a message to a high destination process by first passing it to the trusted low buffer *TLB*, which then gives the message to the communication buffer *CB*. When messages are in the *CB*, the trusted high buffer *THB* signals the high destination process and passes the message to it. Acknowledgments (*ACK*) and negative acknowledgments (*NAK*), and time-outs are used between *THB* and the destination and between *TLB* and the source. These acknowledgments are necessary for reliability and recoverability. They can be exploited as a covert channel because the destination process can modulate its *ACK* and *NAK* messages (or time-outs) to leak sensitive information to low. The Pump itself is trusted and cannot be exploited in this way. Figure 1 shows the Pump.

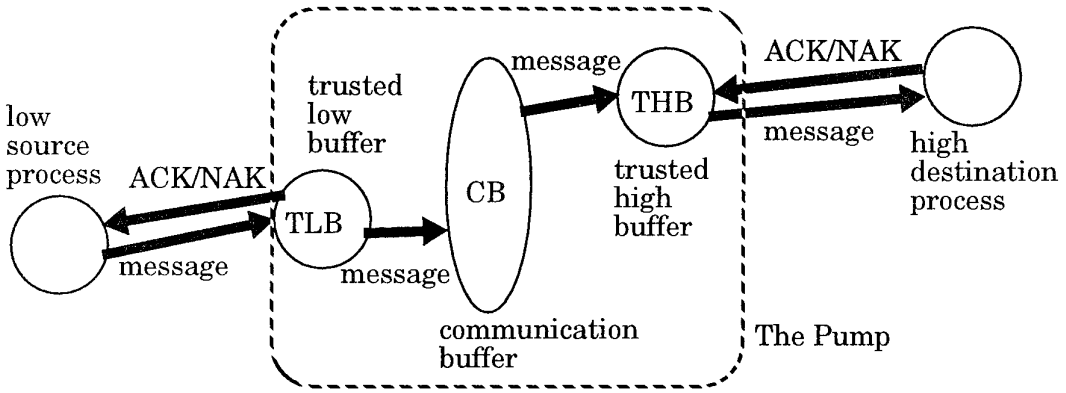


Figure 1. Message Passing From Low to High Using the Pump.

Kang and Moskowitz throttle the covert channel by delaying the acknowledgments from the high destination process in a way that gives approximately the same expected (mean) response time but significantly reduces the influence that the high destination process has on individual response times. The delay is added via a random variable with a modified exponential distribution. By computing a moving average they control the capacity of the channel and further complicate matters for Trojan horses.

The Write-Up Service

Now we look at a write-up service that does not incorporate a covert channel in its mechanism but nevertheless also provides effective reliability and recoverability. Like the pump, our write-up service also depends upon trusted software. The key point of the trust is that trusted software will only send legitimate control messages (i.e. *NAK* is only sent when a write-up fails). Protocol events caused by the write-up service are not due to a Trojan horse. We prevent modulation of the write-up service itself by disconnecting the flow of acknowledgments from high to low, and compensating for this by providing a probabilistic form of guaranteed delivery.

The service provides a set of write-up ports to the low process, that is, the writer. It provides a different set of receive ports to the high process that acts as the destination. The service maintains, in stable storage, a buffer to store the messages. The service follows a fairly conventional protocol, except there is no acknowledgment

from the destination process to the source process:

The buffer slots can be either full or free and a message in the buffer can be removed from a receive-down port or overwritten by the write-up service. The low source process is allowed to query the write-up service regarding the status of a buffer slot, but not the status of a message in the buffer slot. The high destination process can query the status of buffer slots and messages in the buffer slots. The buffer starts with all slots free and no messages in the slots.

1. Low connects to a write-up port.
2. High connects to a receive-down port by specifying the kinds of messages it wants to receive.
3. Low sends a message. If the message is received by the trusted write-up service then an *ACK* is sent to low, the message is placed in a free buffer slot, the slot is marked full, and low may discard its copy of the message. If the message is not received by the write-up service then the trusted write-up service will either send NAK or low will time-out. In either failure case low retries the write-up. If the buffer is full, that is no free buffer slots are available, the write-up service will tell low to wait.
4. The write-up service signals or interrupts the high process to notify it that a message has arrived from low. After either a fixed or random time interval, the message's buffer slot is marked free. Freeing a buffer slot does not remove a message via a receive-down port.
5. High removes the message from its receive-down port. The write-up service does not tell low that the message has been removed from the port. Removing a message does not free its corresponding buffer slot.

To summarize, if we define a message in a free slot as discarded, denoting this condition as *dis*, removed from a receive-down port as *rem*, and overwritten as *over*, we have six possible message conditions:

dis **and not** *rem* **and** *over* (1)

dis **and not** *rem* **and not** *over* (2)

dis **and** *rem* **and** *over* (3)

dis **and** *rem* **and not** *over* (4)

not *dis* **and not** *rem* (5)

not *dis* **and** *rem* (6)

Steps three, four, and five can be repeated until either high or low decides to end the write-up session and disconnects. Flow control can be improved by overlapping several acknowledgments with a sliding window protocol. The low source processes are allowed to know how large the buffer is and when it is full, that is, they can be legitimately blocked when the buffer is full because the state of the buffer does not depend on the destination process. Figure 2 shows the components of the write-up

service.

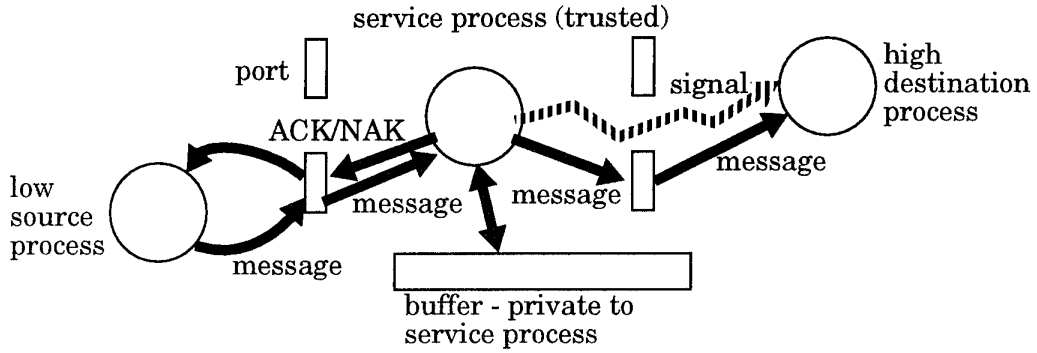


Figure 2. Basic write-up service

This protocol provides communication with conventional flow control between the source and the service process. We could even have an incremental improvement in the overall performance and reliability by having the service process send the messages (instead of a signal) and conduct a separate flow control protocol with the destination, as long as this protocol did not change the rate at which buffer slots were freed by the service process. The flow control is not modified by random extensions of the delay associated with sending a message. There is no covert channel due to acknowledgments sent from the high destination process to the low source process because there are none. If a malicious destination process refuses to receive messages, then the messages are overwritten[1]. Thus performance and security exceed that of the more general Pump mechanism. As we shall show next, the reliability and recoverability can be made arbitrarily good.

Big Buffers

Our write-up service depends on careful buffer management to avoid overwriting a message, condition (1) above. In normal operation and during short term failure, the success of our approach depends on being able to establish a *big* (enough) *buffer*. As we will show, it is possible to determine the probability of overwriting an update projection, as a function of the buffer size and system load. Because of this we can chose a buffer size that makes the buffer practically infinite.

Let us define a catastrophic failure κ as a failure of a database system that causes parts of some transactions to be lost and the database system to produce an incorrect history. This can happen even when correct transaction processing mechanisms are used because the failure (most likely a combination of failures) causes one of the underlying assumptions to be untrue (e.g. hardware failure or single-event upset in the running software). Because of the transaction processing mechanisms and the care taken in designing and implementing the system we expect the probability p_κ of catastrophic failure κ to be relatively small. Now define p_ω as the probability that an update projection will be overwritten. If the size L of the buffer is sufficiently large so

1. We make no claim to protect against denial of service, but such behavior would be detected quickly and the offending software removed.

that $p_\omega < p_K$, then we say the buffer is a big buffer.

How big does a buffer need to be to be a big buffer? To answer this we model the destination process as a server in an M/M/1 queuing model[1], where the queue is finite. Recall that M/M/1 queueing models have exponentially distributed arrival and service rates, a single server, and are used to find steady-state values. The mean arrival rate of the service requests (write-ups) is denoted λ and the mean service rate (removal of messages by the destination process) is denoted μ. We call the ratio λ/μ the *offered load* (imposed on the system) and denote it by a. Offered load a represents the relative load on the system and is measured in units called erlangs. As a concrete example of how offered load a relates to performance we can find the delay for a particular offered load on our write-up system, using Little's Law [9]. Let L be the mean number of requests in a queue or in the server and W the mean length of time it takes request to pass through the system (i.e. sojourn time); then

$$L=\lambda W \tag{7}$$

for a wide range of queueing models, including the M/M/1 model with finite queue size.

For finite queues, the easiest way to apply Little's Law is to calculate the mean number of requests in the queue directly. Queueing theory [3] gives us the probability p_n of n update projections being present in the finite queue as

$$p_n = (1-a)a^n/(1-a^{max+1}) \qquad \text{for } 0 \le n \le max$$

$$p_n = 0 \qquad \text{for } n > max \tag{8}$$

where max is the size of the buffer. We then compute the mean number of requests in the queue as $L = \sum_{0 \le n \le max} n \cdot p_n$.

So, if our write-up system was receiving one update projection per second on the average (i.e. λ=1.00), the buffer size was 600 update projections, and the offered load was a=0.99 erl, then, by Little's Law, a write-up would take roughly two and a half minutes to propagate, on the average. Practical systems operate with much smaller offered loads; for example if we take a=0.5 erl, the update projection propagates in about one second. These values would hold even in an untrusted system that could use conventional flow control protocols.

If we set $n=max$ in equation (8), we get p_{max} the probability of a full buffer. Since a full buffer causes an overwrite, we can treat p_{max} as p_ω probability of overwriting an update projection. Figure 1 shows a plot of buffer size as a function of offered load,

---

1. Besides being tractable, this model is appropriate because the source and destination processes in a replicated architecture are essentially the same, though possibly loaded differently. With respect to tractability, our current model of a finite M/G/1 queue must be run overnight to compute a single data point. Its results tend to agree with the more tractable M/M/1 model.

for a range of overwrite probabilities.

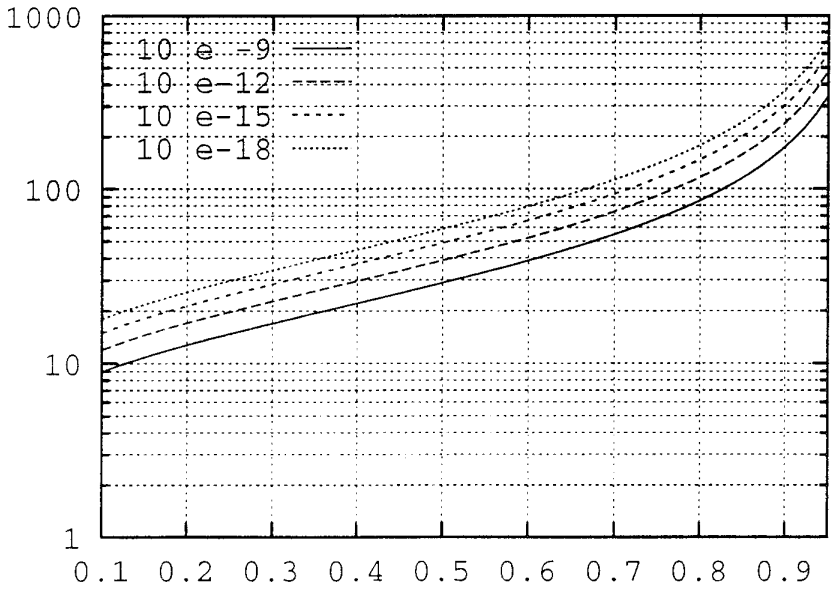


Figure 3. Buffer size as a function of offered load for overwrite probabilities of 10^{-9}, 10^{-12}, 10^{-15}, and 10^{-18}.

As we see from Figure 3, a buffer size between 100 and 500 is sufficient for offered loads as high as a=0.95 erl with a overwrite probability of 10^{-12}, a condition where the average delay for our previous example is about 28 seconds. Since there are 3.1536×10^7 seconds in a year, it is unlikely that our write-up system will have an overwrite during its useful lifetime. Significant increases in reliability can be obtained for relatively small increases in buffer size. If we reduce our overwrite probability to 10^{-15}, we only need a buffer size of about 600 at a offered load of a=0.95 erl.

Since update projections are relatively small objects (an average size of 1K bytes is quite generous for a logical update projection[1]) provision of big buffers is practical. In practice the average size of an update projection is likely to be an order of magnitude smaller. Even if our update projections were 1K bytes, we would only need about 600K bytes of buffer storage for a write-up service.

Because buffer exhaustion cannot be used to communicate, we assume no attempt by the untrusted sender or receiver to fill up the buffer in order to cause an unauthorized information flow. This justifies our use of conventional models based on independent arrival and service times, and conventional steady state values.

---

1. A logical update projection is implemented by sending the text of an update transaction rather than the physical writes it generates.

Recoverability

From the perspective of the source processes and the write-up service proper, the write-up service appears to handle system failures just as a conventional system would. Messages in the buffer are in stable storage; transactional logging procedures can be used to restore the buffer in the event of a system failure. If source processes use similar techniques, they can retransmit messages that were not acknowledged by the write-up service. For this reason, the write-up service provides recoverability for failures of the source processes and of the write-up service itself; we will not discuss it further.

The question of recoverability with respect to failure of the destination process is more interesting. Our basic approach is to make the write-up service buffer large enough to hold all the messages that may be sent before a destination process can recover. Here we are only able to succeed because we restrict the problem to replicated-architecture database systems. Because we are using source processes with finite memory we know we will have to chose to discard some update projections (write-up messages) in the event of a long-term destination process failure. This choice has nothing to do with write-up strategies but rather with the finite capacity of the source process. The source must continue to process new updates at its own security class and it must eventually run out of space to store the new update projections it wishes to propagate and so must discard some of them. This is not a problem; the same choice is made for conventional distributed database systems [2, § 8.5]. For this reason, if we restricted ourselves to use of the Pump, we would still have to discard some write-up projections in the event of long-term failure[1].

Since we know some update projections will have to be discarded in some cases, we can chose to define a short-term failure to be one that fits our desired range of offered loads. That is, if we expect offered load a to be small and failures to be infrequent, we can define "short" as a longer period of time than if we expect frequent failures of the destination process or if we expect offered load a to be relatively large. In any case, in determining the required buffer size, we simply treat short-term failures as additional write-up requests that tie up the system for some period of time equal to the time needed to detect and recover from the failure.

Usability Enhancements To The Basic Service

There are some non-critical enhancements we can make to the basic service to improve its usability in practical systems: message time-outs, overwrite priorities, and variable buffer sizes.

First, we can make it easier for the destination process to manage its rate of message receipt and the write-up service to adjust its buffer size. To do this, we set a timer for each message when it is accepted by write-up service. Messages received for write-up are stored until they either expire or they are received by a high destination process. If a message is received it is marked as such but is not removed from the buffer. Only high destination processes can tell if a message has been received. The time-out

---

1. Recall that one-copy serializability does not require all writes to update all copies.

period for messages is fixed at system generation time. Upon time-out the message expires, is marked as such, and it *may* be overwritten or discarded by the write-up service. An expired message may be received and a received message will expire. Only expired messages are overwritten or discarded. A human user (database administrator or system security officer) can monitor the performance of the destination process with respect to the time-outs and adjust the buffer size of the write-up service if necessary. If we use this option, we want to hide the buffer size from the source process.

A second enhancement we can make will improve the ability of the write-up service to ensure that critical messages are more likely to be received. The write-up service can provide a priority parameter that indicates the criticality of the message. If an overwrite is necessary, the lower priority messages will be overwritten first.

A third enhancement we can make is to provide variable buffer size. The write-up service does not need to maintain a fixed buffer size, if the buffer size is not visible to the sending processes. With this approach, the write-up service would maintain an estimate of offered load a and adjust the buffer size as needed. In the case of a full buffer, the write-up service would first try to expand the buffer and then overwrite an earlier message if no more free space were available. Our model shows that this is possible, since big buffers will fit easily into the stable storage space available on a dedicated frontend or replica controller.

It is also possible to let the source process know the size of the buffer used by the write-up service but still vary the effective buffer size. If the destination process is designed as an interrupt handler (i.e. a small program that quickly removes data from a port and then schedules work for a larger process that uses the data) it can have a variable size buffer. This second buffer can be implemented at the destination security class and thus will be invisible to the source process. The destination process buffer can vary according to a, and the destination process will only be responsible for receiving write-ups from the service. The replicated architecture database

system can then accept update projections from the destination process.

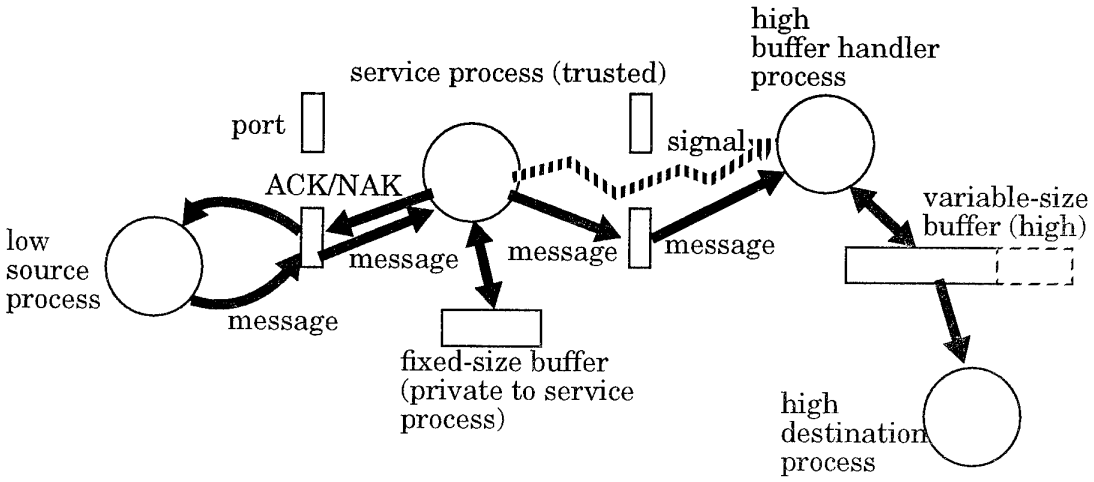


Figure 4. Improved write-up service with variable-size buffer

Conclusions

The Pump represents a good general mechanism for writing up. However, we believe that practical covert-channel-free alternatives exist for the special case of the replicated architecture, with comparable or better time performance without a meaningful sacrifice of reliability and recoverability. The write-up service we have described here is one alternative. Our alternative mechanism has the same performance as an untrusted communication mechanism, that is, no delays are introduced. It has no covert channel due to acknowledgments. There is a nonzero probability of overwriting a message, but the reliability and recoverability can be made arbitrarily good by appropriate choice of buffer size. Acknowledgements and flow control can be extended to cover, separately, both source-to-service and service-to-destination communications. We can easily make the write-up mechanism more reliable than the system it supports. It is not clear that an adjustable covert channel can be set to be smaller than the smallest covert channel that might be exercised, particularly in light of the small message criterion of Moskowitz and Kang. On the other hand, we must admit that our write-up service is not general, but limited to an important special case and may not apply to other special cases.

Stable storage is inexpensive compared to the cost of developing new applications on high-assurance trusted systems. This is the same justification for the replicated-architecture approach, which is the place we expect this service to be used. Where the offered load a is less than 1.1 erl, we can provide the desired reliability within the bounds of conventional disk systems. In the more likely case, where the offered load a is less than 0.95 erl, we can succeed with buffers whose size is between 10^2 and 10^3 update projections.

Our write-up service has not been specified so that it can deal with creation of new data items at higher security classes. This kind of operation is not available in the current SINTRA prototype [7], but is necessary for fully general multilevel transac-

of new data items via blind write-up, but this seems less than satisfactory. Future work should investigate models and mechanisms for extending big buffer write-up to handle creation of new data items in a more elegant fashion. We also plan to look at more advanced models of buffer size, such as M/G/1 queues with finite buffers.

Acknowledgments

Carl Landwehr suggested several improvements to the paper. Our queueing theoretic model (both on paper and in Mathematica) has benefitted from several discussions we had with Ira Moskowitz. Insightful comments by anonymous referees also improved this paper.

References

1. BELL, D. and LAPADULA, L. Secure computer systems: unified exposition and Multics interpretation. MTR-2997, MITRE Corp., Bedford, MA, 1975.
2. BERSTEIN, P., HADZILACOS, V., and GOODMAN, N. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
3. COOPER, R. *Introduction to Queueing Theory*, North-Holland, 1981.
4. COSTICH, O. and JAJODIA, S. Maintaining multilevel transaction atomicity in MLS database systems with kernelized architecture. in *Database Security VI: Status and Prospects*, B. Thuraisingham and C. Landwehr, ed. North-Holland, 1993.
5. COSTICH, O., McLEAN, J., and McDERMOTT, J. Confidentiality in a replicated architecture trusted database system: a formal model. in *Proceedings of the 1994 Workshop on Computer Security Foundations*, Franconia, NH, June 1994.
6. FROSCHER, J. and MEADOWS, K. Achieving a trusted database management system through parallelism. in *Database Security II: Status and Prospects*, C. Landwehr, ed. pp. 151-160, North-Holland, 1989.
7. KANG, M. FROSCHER, McDERMOTT, J., COSTICH, O., and PEYTON, R. Achieving database security through data replication: the SINTRA prototype. NRL TM 5400-041A. February 1994.
8. KANG, M. and MOSKOWITZ, I. A pump for rapid, reliable, secure communication. *1st ACM Conference on Computer and Communications Security*, Fairfax, Virginia, November 1993, pp. 119-29.
9. LITTLE, J. A proof of the queueing formula L=λW. *Operations Res.*, 9, 3, 1961, pp. 383-387.
10. MATHUR, A. and KEEFE, T. The concurrency control and recovery problem for multilevel update transactions in MLS systems. in *Proceedings of the 1993 Workshop on Computer Security Foundations*, pp. 10-23, Franconia, NH, June 1993.
11. FROSCHER, J., KANG, M., MCDERMOTT, J., COSTICH, O., and LANDWEHR, C. A practical approach to high assurance multilevel secure computing service. submitted for publication.
12. MOSKOWITZ, I. and KANG, M. Covert channels - here to stay?, to appear in pro-

ceedings of COMPASS '94.

13. R. SANDHU, R. THOMAS and S. JAJODIA. Supporting timing-channel free computations in multilevel secure object-oriented databases. in *Database Security V: Status and Prospects*, C. Landwehr and S. Jajodia, eds., pp. 297-314, North-Holland, 1992.

14. R. THOMAS and R. SANDHU. Implementing the message filter object-oriented security model without trusted subjects. in *Database Security VI: Status and Prospects*, B. Thuraisingham and C. Landwehr, eds., pp. 15-34, North-Holland, 1991.

Trusted RUBIX: A Multilevel Secure Client-Server DBMS

James P. O'Connor

Infosystems Technology, Inc.
1835 Alexander Bell Drive, Suite 230
Reston, VA 22091

Abstract

In this paper, we present a design for a multilevel secure (MLS) database management system (DBMS) intended to meet the Class B2 requirements of the Department of Defense Trusted Computer System Evaluation Criteria. Our design approach allows us to support client-server operation without introducing trusted code. We also present a discussion of the issues that arise in the development of a multilevel secure client-server DBMS and an analysis of the relationship between client-server architectural design choices and assurance.

1 Introduction

There is a significant requirement for multilevel secure (MLS) database management system (DBMS) technology in the Department of Defense (DOD) and the intelligence community. A multilevel secure DBMS is a DBMS that can store and process information at multiple security levels and serve multiple users some of whom may not be cleared for all the information in the system. The area of multilevel database management has been actively pursued in the research community, and significant progress has been made in the theory and practice of multilevel DBMS design and implementation.

The Client-Server model is fast becoming the predominant model of database access. A client-server architecture offers many advantages over traditional mainframe-based monolithic architectures, among these are: improved access to shared data resources, more effective use of computational resources, support for incremental growth, and support for Open System standards resulting in greater database interoperability. However, multilevel secure database technology and implementation has lagged behind advances in Open Systems distributed computing architectures. The result is that users must sacrifice Open Systems/distributed computing solutions in order to meet security requirements.

In this paper, we report on an effort to design and implement a multilevel secure client-server DBMS intended to satisfy the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) functionality and assurance requirements for a Class B2 computer system [1, 2]. The starting point for this effort was an existing commercial relational DBMS product called RUBIX. An earlier phase of this effort focused on reengineering RUBIX to satisfy the B2 requirements. The results of that phase are described in [3]. This

paper describes progress since then in migrating the standalone version of Trusted RUBIX to a client-server architecture.

In section 2, we provide a brief overview of the standalone version of Trusted RUBIX that was the implementation basis for the client-server version of Trusted RUBIX. In section 3, we discuss multilevel secure client-server DBMS design issues. In section 4, we describe the design for the client-server version of Trusted RUBIX. In section 5, we discuss the relationship between client-server architectural choices and assurance. In section 6, we present conclusions and discuss future research.

2 Overview of Standalone Trusted RUBIX

The starting point for our client-server implementation effort was a standalone version of Trusted RUBIX running on the AT&T 3B2 running UNIX System Laboratories' (USL) System V Release 4.1 ES (SV4.1ES). Since this system forms the core of the server in our design, we provide a brief overview of its policy and architecture.

2.1 Security Policy

The Trusted RUBIX security policy is an adaptation of the policy developed as part of the SeaView project [4]. The mandatory access control (MAC) policy is a straightforward interpretation of the Bell and LaPadula [5] model. Subjects are operating system processes running on behalf of users. Each subject has a security level that is dominated by the corresponding user's clearance. The MAC objects are databases, schemata, relations, indexes, view definitions, access control lists, and tuples. A subject can read an object if its security level dominates the security level of the object. A subject can write an object if its security level is equal to the security level of the object.

Discretionary access control (DAC) is enforced in Trusted RUBIX by allowing users to specify which users and groups are authorized for specific access modes (privileges) on database objects. The DAC objects in Trusted RUBIX are databases, schemata, stored relations, and views. Different modes of access are permitted on different objects (e.g., select, insert, delete, and update are some of the modes supported on tables). A special NULL access mode is used to support the explicit denial of access.

The details of the Trusted RUBIX adaptation and interpretation of the SeaView policy are discussed in [3]. A user-level description of the Trusted RUBIX protection mechanisms is provided in [6].

2.2 System Architecture

The Trusted RUBIX architecture is based on the concept of a *protected subsystem* [7]. All Trusted RUBIX data are stored in one or more volumes, which are single-level operating system objects. To support fine-grained multilevel objects (viz., tuples), labels are attached to individual database items within each operating system object. Note that these labels are DBMS labels not operating system labels—the operating system views these labels strictly as data and attaches no security significance to them. The DBMS is trusted to properly

associate and maintain the label of each item and to correctly interpret those labels so that, in cooperation with the operating system trusted computing base (TCB), the security policy can be correctly enforced. To encapsulate protected data and to implement the DBMS security policy, this architecture employs *trusted subjects.* A trusted subject is a subject that runs with special privilege and can bypass the operating system's security policy whenever this is necessary to implement the DBMS security policy.

Each Trusted RUBIX volume is encapsulated using two mechanisms. The first mechanism is to add the special category RUBIX to the volume security label. Only subjects in the Trusted RUBIX TCB can run with this category in their label. The second mechanism used to protect database volumes is to make them accessible only to the reserved group RUBIXTP. Access to this group can only be obtained when invoking a program in the RUBIX TCB (via the UNIX setgid mechanism). These mechanisms ensure that only subjects in the RUBIX TCB can directly access volume data (i.e., the TCB is *non-bypassable*).

The programs that make up the Trusted RUBIX TCB are also protected by two mechanisms. First, they are MAC protected from unauthorized modification by labeling them with the hierarchical level USER_PUBLIC which is dominated by the level of all untrusted processes. Second, these programs are protected by installing them with execute only permissions. When in execution, these programs are protected by the underlying Trusted UNIX process isolation mechanism. These mechanisms ensure that the TCB is *tamper-resistant.*

The implementation of the Trusted RUBIX TCB employs a technique known as *privilege bracketing.* Privilege bracketing refers to the procedure of explicitly acquiring a privilege immediately before a system call requiring that privilege and releasing the privilege immediately after the call completes. The motivation for privilege bracketing is to minimize the execution time during which a trusted process holds a privilege, thereby supporting the *least privilege* principle within the DBMS TCB.

3 Multilevel Secure Client-Server DBMS Design Issues

There are a large number of design issues that arise in developing an multilevel secure client-server DBMS architecture to satisfy higher assurance levels (e.g., B2). Many of these issues are not security relevant, or also arise in the context of a centralized multilevel secure DBMS. The following are some of the security-relevant architectural issues that are unique to multilevel secure client-server database management systems:

- *Placement of trusted DBMS code on the client machine.* It is desirable to have a design that limits all security relevant DBMS functions to the server. The reason for this is that it is easier to reason about the correctness of a security mechanism that is not distributed between client and server, and therefore, a higher level of assurance in its correctness can be obtained. One difficulty in attaining this limitation is that it requires a relatively sophisticated secure distributed computing infrastructure (SDCI) to implement. Another difficulty is that it imposes limits on client functionality (e.g., support for trusted applications).

- *DBMS or OS/network identification and authentication.* In a client-server DBMS, there must be some way for the server to identify and authenticate the user. This function can be performed by the DBMS itself (e.g., by sending a user identifier and password to the server) or the DBMS can utilize the underlying SDCI in such a way that DBMS-based authentication is unnecessary. The problem with the first approach is that it requires that some portion of the client DBMS be trusted. This is a consequence of the trusted path requirement that is introduced at B2. The problem with the second approach is that it requires a relatively sophisticated SDCI.

- *DBMS or network listener.* A client-server DBMS ultimately requires that there be some software that listens to a well-known address on the network and responds to client requests. In a process-per-user server architecture, this listener is generally a separate process which will start up an instance of the server on demand. In a multithreaded architecture, the listener is generally a standing instance of the server. It is desirable that this function be performed in the SDCI for two reasons. First, the listener must be multilevel[1] and therefore would increase the size of the DBMS TCB. Second, the listener is responsible for reliably associating a user-id with a request and therefore requires a trusted counterpart on the client platform. If this function is performed by the DBMS, then there must also be trusted DBMS functionality on the client platform. Relying on the SDCI for this function has the same drawbacks mentioned in the previous two items.

- *Single-level or multilevel client platforms.* It would be desirable to support an architecture with multilevel servers and single-level client platforms. The problem with such an architecture is that the client platform must be relied upon in some way for identification and authentication (except in the case of a single-level component dedicated to a single user). If this identification is to be used for MAC purposes, then the client platform must be at least as assured as the server (viz., B2). One possibility is to use single-level client platforms evaluated at the C2 level and use their I&A only for DAC purposes. Architectures such as this can use the identification and authentication inherent in communications security (COMSEC) lines connecting the client and server platforms to satisfy the mandatory trusted path requirement. This approach is consistent with the ideas underlying the Trusted Network Interpretation of the TCSEC [8].

- *Homogeneous or heterogeneous client platforms.* It is preferable for a client-server DBMS to run on heterogeneous client and server platforms. The problem with this is that the required SDCI is not widely available at a high level of assurance. The most desirable solution to this problem is to have multiple vendors develop support for these services from a service and protocol definition that has been adopted as an industry standard. This is being done in the CMW world with the MaxSix architecture. Another possibility is to have a third party provide the services as an additional

[1] It is possible to have multiple single-level listener processes, but this would require having one active listener for each point in the lattice.

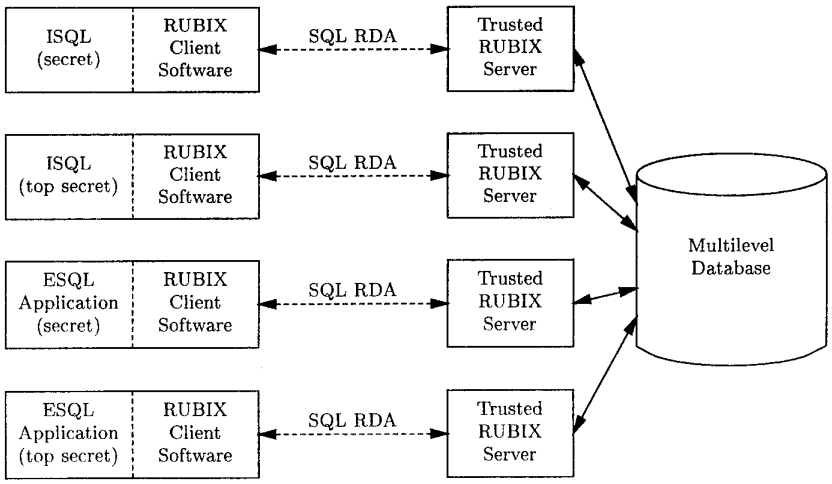


Figure 1: Trusted RUBIX Client-Server Architecture.

trusted layer implemented on top of the heterogeneous platforms as is done in ORA'a Theta [9].

- *Single or multiple administrative domains.* It is also desirable for a client-server DBMS to allow clients and servers to have different user-id, group-id, and security level spaces. The disadvantage of this is that there must be functionality to do mapping between these attributes. There is also the issue of who does the attribute mapping—the DBMS or the SDCI. This is an important decision because this is clearly a security relevant function. Having a single administrative domain removes this problem, but the result is an architecture that is inflexible and not scalable.

The Trusted RUBIX resolution of these issues will be covered in the next section.

4 Trusted RUBIX Client-Server Architecture

The system architecture for Trusted RUBIX is shown in Figure 1. Clients are untrusted application programs that have been linked with the Trusted RUBIX client software so that they can communicate with the Trusted RUBIX server. There is one instantiation of the server for each active client. A given client and server pair can run on the same machine or on different machines connected on a network. The server must reside on the same machine as the data to be accessed. All communication between client and server takes place using the SQL Remote Database Access standard protocol [10, 11]. A client can access multiple servers concurrently (although no mechanism for distributed transaction management is provided). Currently, two types of clients are supported: the standard Interactive SQL (ISQL) interface and user-developed Embedded SQL (ESQL) applications.

Our design approach is to layer this architecture onto an existing SDCI in such a way

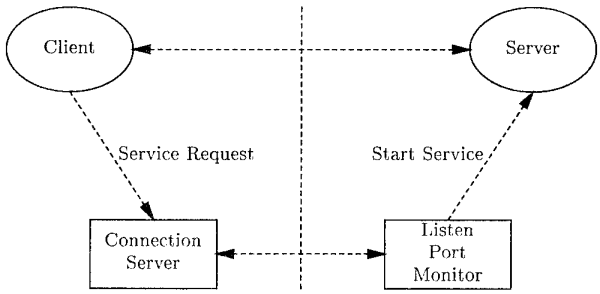


Figure 2: SV4.1ES Secure Networking Facilities

that *no* additional trusted code is introduced. To make this possible, the SDCI must satisfy the following requirements:

- *Remote Process Invocation.* The SDCI must provide a way to invoke a process on a remote machine at the invoker's current level, user-id, and group-id.
- *Single-level connection establishment.* The SDCI must provide a way to establish a single-level connection between the invoker and the new process.
- *Network security policy enforcement.* The SDCI must ensure that the services do not violate intra- or inter-host mandatory or discretionary access controls.
- *Data Confidentiality.* The SDCI must protect the confidentiality of transmitted data (this can be accomplished by physically protecting the transmission media).
- *Attribute Mapping.* The SDCI must perform user-id, group-id, and security level mapping.

The SDCI we selected for the implementation of Trusted RUBIX was the secure networking services of SV4.1ES [12]. The primary components of the SV4.1ES secure networking facilities are the connection server and listen port monitor (Figure 2). The connection server handles all client-side connection establishment as a single service. The listen port monitor is a daemon that listens on the server machine for incoming connection requests, accepts the requests, and starts services that have been requested. In order for these facilities to be used to provide a service (e.g., Trusted RUBIX), the service must have been previously registered with the connection server on the client machine and the listen port monitor on the server machine.

When an application needs to access a service on a remote machine, it makes a request to the connection server. The connection server validates that the connection to the remote machine is permitted at the application's level. If the connection is permitted, the connection server opens a connection to the corresponding listen port monitor on the remote machine. Once the connection is established, the connection server and listen port monitor exercise a mutual authentication scheme (a cryptographic scheme based on secret keys). If the authentication succeeds, the connection server passes the client's user-id, group-id, and

security level to the port monitor. The port monitor maps the user's identity to a local user identity and starts the server corresponding to the requested service. The connection server then passes its end of the connection to the client, and the port monitor passes its end of the connection to the newly invoked server. The client and server are now connected at the desired security level and user-id and can begin a session.

The details of how Trusted RUBIX utilizes these facilities is addressed in the following sections.

4.1 Detailed Design

We describe the design of client-server extensions to Trusted RUBIX in terms of its module structure, internal layering, and its process structure. The *module structure* is a decomposition of the system into modules. Each module consists of a set of closely related procedures. The following are the modules related to the client-server portion of the Trusted RUBIX architecture (the structure of the server engine itself is beyond the scope of this paper).

- The Interactive SQL (ISQL) module provides an interactive SQL interface. This interface can be used to submit ad hoc queries to a Trusted RUBIX server.
- The Embedded SQL Preprocessor (ESQLP) module is a preprocessor that takes a C program with embedded SQL statements and translates it into an C program that can be compiled and linked with Trusted RUBIX client software to access the server.
- An ESQL Application (ESQLA) module is an application module generated by the ESQL preprocessor.
- The SQL Client Interface (SQLCI) module provides a call-level interface that can be used to access the Trusted RUBIX server. There are two versions of this module, a network version and a standalone version. The network version uses the RDA module to access the server. The standalone version uses the RXIPC interface directly and is therefore only usable when client and server are on the same machine.
- The Remote Database Access (RDA) module provides low-level services that allow a client to start/terminate a remote server, manage transactions, execute SQL queries on the server, and retrieve query results and status information. This module utilizes the SQL Remote Database Access protocol to support client-server communication. This module supports both a client-side and server-side interface.
- The Secure Networking Facilities (SNF) module provides services required for invoking the server on a host and establishing a secure communication channel between client and server. This module supports both a client-side and server-side interface. This module is not trusted in itself, but hides the details of the underlying secure networking facilities. When these facilities change, only this module needs to be modified.
- The Server Driver (SD) module is the server side controller for Trusted RUBIX. It accepts requests over the network connection (via RDA) and executes them on the

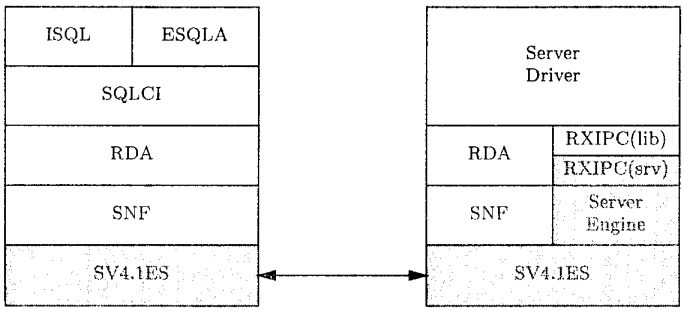


Figure 3: Trusted RUBIX Software Architecture

Server Engine (via the interface provided by RXIPC). Query results and status information are then returned to the client.

- The RUBIX Interprocess Communication (RXIPC) module hides the details of the IPC interface to the Trusted RUBIX engine. The Trusted RUBIX engine runs isolated in a separate address space (as described in Section 2) and is accessible only through this interface. Functions on the interface of this module implement a form of remote procedure call by translating calls into IPC requests to the rxserver process (which is also encapsulated within this module).

- Server Interface (SI) module provides a function call interface to the server. This can only be accessed through the RXIPC module since direct linking with this code would result in a non-isolated TCB.

It is important to note that of the above modules, only the RXIPC and SI are within the RUBIX TCB; and these modules are required even in the standalone architecture. That is, no new trusted code was introduced to support client-server operation.

The layering of the Trusted RUBIX client-server modules is shown in Figure 3. The shaded part of the figure indicates modules in the Trusted RUBIX TCB. The RXIPC module is unique in that is consists of a trusted and an untrusted part. These parts correspond to the two ends of the IPC connection. Note that the "Server Engine" module in the figure actually consists of a large number of modules (including the Server Interface module) which are not shown in this diagram. A discussion of these modules is beyond the scope of this paper.

The *process structure* is a decomposition of the run time activities of the system into processes. Trusted RUBIX actually has two process structures corresponding to the two possible builds of the system (i.e., the standalone process structure and the client-server process structure). In the following, we will focus on the client-server process structure.

As shown in Figure 4, the process structure for Trusted RUBIX consists of three processes. There is the application process which consists of an application linked with the Trusted RUBIX client code. The application can be either the ISQL application supplied with Trusted RUBIX or an application a user developed using ESQL. This process runs on behalf of the user and has no special privileges.

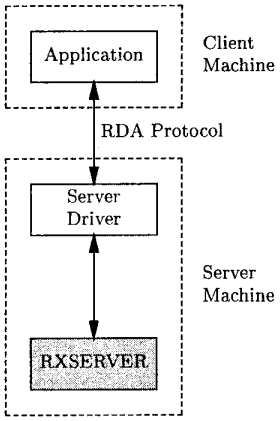


Figure 4: Trusted RUBIX Process Structure

The server driver process runs on the server and reads database requests from the application process (using the RDA protocol) and sends those requests to the rxserver process using the RXIPC interface. This process is invoked as the result of a remote process invocation request by the client (this functionality is supplied through SNF). The process is invoked at the client's level, user-id, and group-id, and runs with no special privilege. As part of the invocation process, this process obtains a single level connection to the application process.

The rxserver process forms the Trusted RUBIX TCB (this is indicated by the shading in Figure 4). It reads database requests from the server driver and executes them against the requested database. The rxserver process performs all the access mediation functions of Trusted RUBIX. It is started by the server driver when the server driver receives a request to open a database. Rxserver is a trusted subject that runs under the user's user-id (the group-id of this process is set to RUBIXTP so that it can access protected data).

4.2 Operational Scenario

The operational characteristics of this architecture are best illustrated with a scenario. A user starts a session by logging in to a client machine at a specific level. Since the client machine is a B2 platform, it identifies and authenticates the user via a trusted path. Once authenticated, the user can invoke a Trusted RUBIX application on the client machine. When the application requests a connection to the server (e.g., through an SQL CONNECT statement), the request is ultimately translated into request to the SV4.1ES networking facilities to access a remote service (viz., the Trusted RUBIX server). The networking facilities first validate that the requested invocation does not violate network security policy. If the operation is permitted, the networking facilities invoke a Trusted RUBIX server driver process on the remote host at the application's security level and using the corresponding user's user-id, and group-id (possibly mapped). The final step in the invocation process is to provide the client application and server driver with a single-level connection over which

database access requests and results can be passed.

The server driver is the client's agent on the server machine. When the client requests access to a database, the server driver invokes the rxserver process. This process inherits the security level and user attributes of the server driver. An interprocess communication channel is set up between the server driver and the rxserver process for the communication of requests and results.

The client sends queries to the server (server driver) using the RDA protocol. These queries are forwarded to the rxserver process which mediates access based on the user's level and user-id. Since the server driver is untrusted, the rxserver process (i.e. the Trusted RUBIX TCB) labels all data coming from the server driver at that level, and mediates all database accesses based on that level. Since the rxserver process was started under the user's user-id (possibly mapped), it has a basis for making discretionary access control decisions. Results are sent from the rxserver process to the server driver and back to the client application. When the client terminates its connection, the server driver notifies the rxserver process to terminate and then terminates itself.

In this section, we presented a design for a multilevel secure client-server DBMS. Our basic design approach was to layer the DBMS client and server onto an existing secure distributed computing infrastructure. This approach allowed us to transition Trusted RUBIX to a client-server architecture without the introduction of any new trusted code. Another notable aspect of our design is that all DBMS policy enforcement functions are centralized on the server (i.e., not distributed). This is important because it makes it easier to reason about the correctness of the mechanism enforcing the system security policy.

5 Assurance in Client-Server Architectures

Assurance is concerned with providing convincing and rigorous technical arguments that the security mechanisms in a secure system are implemented correctly. The assurance arguments for distributed systems are inherently more complex than the corresponding arguments for centralized systems. This section discusses the relationship between certain critical client-server DBMS design choices and assurance. In this discussion, we will limit ourselves to architectures where the DBMS code is layered on top of an existing distributed computing infrastructure such as in the design presented in the previous section.

To build a client-server DBMS on top of a secure distributed computing infrastructure is to build on top of a distributed TCB where each host platform (client or server) forms a partition of that TCB. We will use the terminology from the Trusted Network Interpretation of the TCSEC [8] and refer to these partitions as Network TCB (NTCB) partitions. Figure 5 illustrates the layering of the DBMS client and server on top of the distributed TCB. The degree of assurance that can be attained with such an architecture depends heavily on the characteristics of the client and server components of the DBMS software. To aid in our analysis, we categorize client and server components based on their policy-related responsibilities. Our two categories are: support for DBMS mandatory access control and support for DBMS discretionary access control.[2] A given client-server architecture can now

[2]The notion of supporting policies (viz., identification, authentication, and audit) is rolled into these



Figure 5: Client-Server Distributed TCB

be characterized by assigning zero or more of the categories to the client and to the server. The result is a set of 16 possible assignments (4 possible combinations for client and server).[3]

Certain (overlapping) classes of the above assignments are of particular interest from an assurance point of view and are discussed below.

- *No DBMS policy enforcing software in the client or server.* This would be the case in an architecture that relied completely on the underlying distributed TCB for its security (similar to a Hinke/Scheafer approach [13] on a distributed TCB). This class offers the highest level of assurance since the DBMS components enforce no security policy in themselves and run with no special privilege with respect to the underlying distributed TCB. Their security characteristics depend completely on the security characteristics of the underlying distributed TCB.

- *Discretionary policy enforcing software in the server only.* This would be the case in an architecture that relied on an underlying distributed TCB for MAC enforcement, but implemented its own DAC on top of the DAC policy of the underlying TCB (similar to a SeaView [14] approach on a distributed TCB). This case would have a high degree of assurance for MAC because the DBMS runs with no special privilege with respect to the underlying distributed TCB. The level of assurance obtainable for DAC would be commensurate with the assurance techniques applied to the DBMS server. The important point here is that effort involved in obtaining a given degree of assurance for this architecture would be no more than that required for a similar standalone architecture. This is because the DAC policy enforcement mechanisms are centralized on the server.

- *Mandatory policy enforcing software in the server.* This would be the case in a trusted subject DBMS architecture where part of the server runs as a trusted subject with respect to the underlying distributed TCB. This is the Trusted RUBIX case. In this case, more analysis is required to gain a high level of assurance because the DBMS server runs with special privilege with respect to the underlying distributed TCB. The difficulty that arises is that the DBMS server constitutes a modification to the underlying TCB, and therefore, any assurance arguments must not only consider the DBMS TCB itself, but also its impact and interactions with the underlying distributed TCB. This is similar to a single machine case, where the assurance arguments for a trusted subject DBMS must extend to the underlying operating system. It needs

categories since these policies can be supportive of MAC, DAC, or both. That is, if a component implements supporting policies for MAC, we consider it a MAC component.

[3]A more detailed analysis could be done using the categories presented in Appendix A of the TNI. The four categories presented there (M,D,I, and A) would yield a total of 256 combinations.

to be determined if these impacts can be isolated to the NTCB partition, or if the assurance arguments would need to extend to the system level. This architecture retains the advantage that the DBMS policy enforcement mechanisms are centralized on the server.

- *Policy enforcing software in the client and server.* In this case, the assurance argument may or may not have to extend into the distributed TCB depending on the approach to MAC enforcement (i.e., TCB subset or trusted subject). But since the DBMS policy enforcement responsibilities are distributed, the assurance argument for the DBMS TCB will be much more complex than in the case when the DBMS TCB is centralized.

This discussion covered only a few of the possible architectures. The important point to remember is that the complexity of the assurance argument for a client-server DBMS architecture (which is directly related to evaluation difficulty) seems to be even more sensitive to architectural choices than are assurance arguments for standalone DBMS architectures.

6 Conclusions and Future Research

In this paper, we presented a design for a multilevel secure client-server DBMS intended to satisfy the TCSEC requirements for a Class B2 computer system. We also presented an analysis of the relationship between client-server design choices and assurance. The major conclusions from this effort are:

- It is possible to develop a multilevel secure client-server DBMS without introducing additional trusted code over that used in the server engine. This can be accomplished by building on an existing secure distributed computing infrastructure. This general approach can be used with servers having trusted subject or TCB subset architectures.
- The complexity of the assurance argument for a client-server DBMS (and therefore the evaluation difficulty) is dependent primarily on two factors: whether the client or server require trusted subjects, and whether the policy enforcement functions of the DBMS are distributed between client and server.
- The enabling technology for a multilevel secure client-server DBMS developed using this approach is not yet mature. Some operating system vendors (e.g., UNIX System Laboratories) provide the necessary functionality, but the functionality is not currently in their evaluated configuration.

There are a number of directions in which this work could be extended. One promising area is to investigate how this architecture can be used with single-level client machines as discussed in section 3. Another possible extension of this work is to allow clients to execute transactions that span multiple servers. If clients are assumed to be single-level, then they may only connect to servers at that single level. Since the connections are single level, and the servers to which they are connected are multilevel trusted, this class of transaction

would not appear to introduce any significant covert channels [15]. This of course assumes that the concurrency control mechanisms of the underlying servers are secure.

A more ambitious extension of this work is to modify the server to support a true distributed database capability. Issues that could be investigated in such an effort include secure distributed transaction management, secure distributed query processing, secure access to heterogeneous databases, security implications of data replication, secure distributed database infrastructure, and impacts of security on Open Systems standards.

Acknowledgements

This work was supported by the U.S. Air Force, Rome Laboratories, under contract F30602-90-C-0045.

References

[1] Department of Defense. Department of defense trusted computer system evaluation criteria. DOD Standard 5200.28-STD, Department of Defense, December 1985.

[2] National Computer Security Center. Trusted database interpretation of the trusted computer system evaluation criteria. Technical Report NCSC-TG-021, National Computer Security Center, April 1991.

[3] C. Testa, B. Wilner, and V. D. Gligor. Trusted RUBIX architecture and policy model interpretation. In *Proceedings of the 8th Aerospace Computer Security Conference*, December 1992.

[4] T. Lunt, D. Denning, R. Shell, W. Shockley, and M. Heckman. The SeaView security model. *IEEE Transactions on Software Engineering*, June 1990.

[5] D. Bell and L. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE Corporation, July 1975.

[6] Infosystems Technology, Inc. *Trusted RUBIX Version 2.0 Security Features User's Guide*, March 1994.

[7] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), September 1975.

[8] National Computer Security Center. Trusted network interpretation of the trusted computer system evaluation criteria. Technical Report NCSC-TG-005, National Computer Security Center, July 1987.

[9] ORA Corporation. An introduction to Theta. Technical Report TM-93-0004, ORA Corporation, February 1993.

[10] International Organization for Standardization. Remote database access – part 1: Generic model, service, and protocol. Draft International Standard ISO/IEC DIS 9579-1, International Organization for Standardization, 1991.

[11] International Organization for Standardization. Remote database access – part 2: Sql specialization. Draft International Standard ISO/IEC DIS 9579-2, International Organization for Standardization, 1991.

[12] Unix System Laboratories. *System V Release 4.1 ES Network User's and Administrator's Guide*, 1991.

[13] T. Hinke and M. Schaefer. Secure data management system. Technical Report TR-75-266, System Development Corporation, November 1975.

[14] T. Lunt and D. Hsieh. The SeaView secure database system: A progress report. In *Proceedings of the European Symposium on Research on Computer Security*, Toulouse, France, October 1990.

[15] S. Jajodia and C. McCollum. Using 2-phase commit for crash recovery in federated multilevel secure database management systems. In *Proceedings of the 3th IFIP Working Conference on Dependable Computing for Critical Applications*, Mondello, Italy, September 1992.

Inference analysis and control:

Chair: E. Gudes

Ben-Gurion Uni., Israel

A Practical Formalism for Imprecise Inference Control

John Hale Jody Threet Sujeet Shenoi*
Department of Mathematical and Computer Sciences
Keplinger Hall, University of Tulsa
Tulsa, Oklahoma 74104-3189, USA

Abstract

This paper describes a powerful, yet practical, formalism for modeling and controlling imprecise FD-based inference in relational database systems. The formalism provides a canonical representation of inference which unifies precise inference and the primitive imprecise inference mechanisms of abduction and partial deduction. Whereas other imprecise (partial) inference models estimate the probability of making inferences, the formalism supports the analysis of the actual imprecise values inferred in a database extension. Imprecise inference is analyzed by transforming a precise database augmented with additional "catalytic" relations, conveying possibly imprecise *a priori* knowledge, into an equivalent imprecise database. The analysis of imprecise inference and the related inference control methodology are highly flexible and robust. They can be directly applied to classical, MLS, and imprecise databases. With minimal modifications, they also can be used in knowledge discovery or database mining.

1 Introduction

Controlling user access to database data has been the focus of considerable attention in the security community. However, serious security compromises also can arise from *inference attacks* [2]. An inference attack occurs in a multilevel secure (MLS) database [11] when a low user is able to infer sensitive information from common knowledge and authorized query responses. Su and Oszoyoglu [19,20] showed that integrity constraints, particularly FDs and MVDs, pose serious security threats. They devised an algorithm for optimally upgrading information to eliminate inference compromises; they also showed that this problem is NP-complete. More recently, Lunt and colleagues at SRI International [5,15,18] have developed an interactive tool, DISSECT, for detecting and eliminating compositional inference channels due to foreign key FDs. The DISSECT model builds on earlier work on inference control, including tools and techniques developed by Buczkowski [1], Thuraisingham [23,24], and Hinke [9]. The current version of DISSECT [18] is limited to analyzing MLS database schemas (intensions) rather than actual MLS relations (extensions). Nevertheless, its success shows that it is possible to develop practical tools for dealing with the difficult problem of inference control.

Effectively controlling imprecise inference in MLS databases remains a relatively unexplored problem of great importance. An imprecise or partial inference compromise occurs when a low user is able to infer an exact value or a set of possible values – an information chunk – for a sensitive attribute with a certain probability. The granularity of the inferred chunk may be small enough and/or its probability high enough to constitute a security breach. Not only is imprecise inference just as damaging as precise inference, it is also far more prevalent.

Several researchers have recognized the importance of controlling imprecise or partial inference in MLS databases. Su and Oszoyoglu [19,20] noted that compromise elimination techniques involving FDs and MVDs

*To whom correspondence should be addressed.
Research supported by NSF Grant IRI-9110709 and OCAST Grant AR2-002.

could be extended to consider ranges of values. Morgenstern [13,14] was among the earliest researchers to formally investigate imprecise inference. He viewed inference as a means to "localize the space of possible values." A unique feature of his approach is its use of constraint expressions involving "spheres of influence" to capture logical inference and an information-theoretic (entropy) measure for imprecise information. The inference control tool developed by Buczkowski [1] uses Bayesian probability to estimate security risks due to imprecise inference; his model extends Morgenstern's framework to permit the propagation of imprecise inference. Garvey and Lunt [4,6,10] have characterized inference channels, including partial inference channels due to abduction and probabilistic reasoning. Although the current version of their DISSECT tool is geared for precise inference, plans are underway to extend it to partial inference control [15,18].

This paper describes a powerful, yet practical, formalism for modeling and controlling FD-based imprecise inference in relational databases. The formalism uses scalar domain partitions – which we call "contexts" – to model imprecise inference and to express integrity constraints [3,16,17]. Contexts also provide a canonical representation which unifies precise inference and the primitive imprecise inference mechanisms of abduction and partial deduction [7,21,22]. Morgenstern's sphere of influence notion for inferred information [13,14] has motivated the development of our formalism. Indeed, the equivalence classes contained in contexts can be viewed as specifying the "maximal" spheres of influence for data involved in imprecise inference. Whereas other techniques (e.g., [1,4,6,10]) estimate the *probability* with which sensitive values are inferred, the context-based formalism enables us to consider the actual "information chunks" inferred in a database extension. To distinguish it from probability-based models, we use the term "imprecise" rather than "partial" to characterize the inference model.

The imprecise inference model is highly flexible and robust. Imprecise inference is analyzed by transforming a precise database, possibly augmented with additional "catalytic" relations [8] conveying *a priori* knowledge available to low users, into an equivalent imprecise database. This technique, which is similar to those used in deductive databases, is particularly suited to modeling the propagation of compromising imprecise inferences in MLS databases. Furthermore, the canonical representation of imprecise inference enables the inference analysis methodology to be directly applied to classical, MLS, or imprecise databases. With minimal modifications, it also can be used in knowledge discovery or database mining.

This paper is organized into six main sections. Following these introductory remarks, Section 2 examines the primitive FD-based inference mechanisms of abduction and partial deduction and provides a foundation for their unification. Section 3 formally defines the context model, an imprecise data model representing our view of imprecise inference. The key concepts of an imprecise FD and an imprecise inference channel are presented in Section 4. Section 5 describes the related imprecise inference control methodology. The conclusions, relevance and applications of the imprecise inference analysis and control methodologies are summarized in Section 6.

2 Imprecise Inference

In general, an FD-based inference may be *deductive*, involving forward reasoning with an FD, or *abductive*, involving backward reasoning. Furthermore, FD-based inference may be precise or imprecise, depending on the nature of the FD and the data used in inference. An imprecise inference compromise occurs when a user can infer a range of values for a sensitive attribute using other information in the database. The granularity of the inferred information could be fine enough to constitute a security threat.

Abduction and partial deduction are the two primary mechanisms for imprecise inference. This section describes the two mechanisms and shows how they can be unified using the notion of an *induced set FD*, a special kind of *imprecise FD*. The unification yields an elegant definition of an *imprecise inference channel* and greatly simplifies the related inference analysis.

Note that the definitions and illustrative examples in this section and in the remainder of this paper use classical relations to simplify the presentation. The extension to MLS relational databases – even those containing polyinstantiated data – is accomplished by imposing the mandatory security constraints on data classifications and user clearance levels. Views of MLS relations created at specific user clearance levels are individually analyzed for potentially compromising imprecise inferences.

2.1 Abduction

The existence of an FD: $X \rightarrow Y$ implies that a precise value for X determines a unique (and precise) value for Y. It is also possible to use the FD – actually its manifestation in a relation – to reason backwards, i.e., to determine a value for X given a precise (or imprecise) value for Y. If the FD corresponds to a many-to-one function, then the inferred value for X is imprecise. This mechanism is commonly referred to as abduction. For example, in the relation below, given the salary $20K$ and the fact that the FD: $Name \rightarrow Salary$ exists, it is possible to determine a set of names, $\{Bill, Bob\}$, that map to the salary value. Note that it is not necessary for a user to know the complete FD mapping. Information is abduced using only the portion of the FD manifested in the relational extension.

Name	*Salary*
Bill	$20K$
Bob	$20K$
Joe	$25K$
Jill	$35K$

To understand how abduction can lead to a potential security compromise, consider the relation above, where salary information is sensitive to the point that low users should not obtain the salary of any employee to any degree of precision. Now assume that the two relations below are accessible to low users (e.g., in a poorly-designed database view):

Name	*Tax*
Bill	10%
Bob	10%
Joe	10%
Jill	15%

Salary	*Tax*
$20K$	10%
$25K$	10%
$35K$	15%

Provided with access to the relations $R(Name, Tax)$ and $R(Salary, Tax)$ above, a low user could infer the following relation by abduction.

Name	*Salary*
Bill	$\{20K, 25K\}$
Bob	$\{20K, 25K\}$
Joe	$\{20K, 25K\}$
Jill	$\{35K\}$

Note that, although the information in the abduced relation is imprecise, a security compromise exists because the low level user has a reasonably good idea of how much each employee earns. The solution, of course, is to classify the Tax attribute in $R(Name, Tax)$ as sensitive so that the abductive channel is closed.

2.2 Partial Deduction

Partial deduction generalizes precise deductive inference. Given an FD: $X_1 ... X_n \rightarrow Y$, partial deduction occurs when values for a subset of left-hand side attributes are used to determine a right-hand side attribute value. Partial deduction using a proper subset of X attributes yields an imprecise Y value.

Job	*Experience*	*Salary*
Technician	*Little*	$15K$
Technician	*Lots*	$20K$
Scientist	*Little*	$20K$
Scientist	*Lots*	$25K$
Manager	*Little*	$25K$
Manager	*Lots*	$35K$

To understand partial deduction, assume that the FD: *Job*, *Experience* $\rightarrow$ *Salary* holds in the relation above. Projecting out the *Experience* attribute yields the following relation.

Job	*Salary*
Technician	$15K$
Technician	$20K$
Scientist	$20K$
Scientist	$25K$
Manager	$25K$
Manager	$35K$

Information in the projected relation above can be used with the FD: *Job*, *Experience* $\rightarrow$ *Salary* in partial deductive inference. This gives rise to the relation presented below. Note that knowledge of a *Job* value allows the inference of an approximate salary, e.g., $\{15K, 20K\}$ for a *Technician*, which may be potentially compromising.

Job	*Salary*
Technician	$\{15K, 20K\}$
Scientist	$\{20K, 25K\}$
Manager	$\{25K, 35K\}$

2.3 Unifying Abduction and Partial Deduction

The abduction example shows how a low user can infer a range or set of possible salaries given values for the right-hand side of the FD. Likewise, the partial deduction example shows how a range of possible salaries may be inferred from a job description, i.e., a proper subset of the left-hand side attributes. In both examples, precise values are used to produce imprecise inferences. In general, however, inference can initiate with precise or imprecise values; using imprecise information gives rise to inferred information which is correspondingly imprecise. The ability to infer precise or imprecise information by abduction and partial deduction stems from "induced set functions" generated from FD mappings manifested in database relations (extensions) [7,21].

Induced set functions for some function $f : X \rightarrow Y$ define the *image* of each $A \subseteq dom(X)$ and the *inverse image* of each $B \subseteq dom(Y)$. The induced set function concept allows us to define *induced FDs* which unify abduction and partial deduction.

Definition: Let $f : X \rightarrow Y$ denote an FD, then $F : X \rightarrow Y$ is the *induced FD* of f such that the *image* of $A \subseteq dom(X)$ is $F(A) = \{y \in dom(Y) : y = f(x)$ for some $x \in A\}$.

As explained above, partial deduction uses values for a subset of the left-hand side attributes in an FD: $X \rightarrow Y$. Let us denote the subset of attributes and the tuple component values corresponding to these attributes by

X' and x, respectively. Partial deduction uses the x value to derive a value $A = \{t[X] : t[X'] = x\}$. The image of A is a set of tuple components in Y which corresponds to the imprecise information that can be inferred from x by partial deduction.

Definition: Let $f : X \rightarrow Y$ denote an FD then $F^{-1} : Y \rightarrow X$ is the *inverse induced FD* of f such that the *inverse image* of $B \subseteq dom(Y)$ is $F^{-1}(B) = \{x \in dom(X) : f(x) \in B\}$.

Abduction uses values for the right-hand side attribute in an FD: $X \rightarrow Y$. (Note that it is necessary only to consider FDs with single attributes on their right-hand sides.) Let us denote the tuple component in Y as y and let $B = \{y\}$. The inverse image of B is the information that can be inferred from y by abduction.

The two definitions above show that an induced FD: $X \rightarrow Y$ is a function mapping sets of X tuple components (subsets of $dom(X)$) to sets of Y tuple components (subsets of $dom(Y)$). This mapping is induced from a relational extension. Induced FDs and inverse induced FDs can be treated in a uniform manner. For this reason, we refer to them collectively as induced FDs. This notion unifies the primitive imprecise inference mechanisms of abduction and partial deduction.

In general, an *imprecise FD* maps set values to set values. (A set and its subsets are considered to be equivalent when determining the functional relationship.) The *induced FD* for a given relation is the "minimal" imprecise FD generated in the relation. To understand the distinction between the two concepts consider the following precise relation.

X	Y
a	1
a	2
b	2

The FD: $X \rightarrow Y$ does not hold in the above relation. However, it is possible to "induce" an FD from X to Y by "merging" precise tuples. The induced FD: $X \rightarrow Y$ holds in the imprecise relation below (left). It is obtained by merging the fewest tuples in the precise relation so that a functional relationship exists between set values.

X	Y
$\{a\}$	$\{1,2\}$
$\{b\}$	$\{2\}$

X	Y
$\{a\}$	$\{1,2\}$
$\{b\}$	$\{1,2\}$

A "coarser" imprecise FD: $X \rightarrow Y$ holds in the relation on the right. The imprecise relation corresponding to this imprecise FD is obtained by merging and "coarsening" (or "clouding") tuples to a greater degree than is required to produce the induced FD. For any relational extension exactly one induced FD can be generated between any two sets of attributes; however, numerous imprecise FDs can be made to hold by appropriately clouding the extension. The induced FD and imprecise FD concepts will be formalized in Section 4 using the notion of a "context" defined below.

3 Contexts and Imprecise Relations

Imprecise inference analysis involves the transformation of a precise database to an imprecise database to materialize potentially compromising imprecise inferences. *Integrity constraints* must be defined for the relations containing imprecise data (set-valued tuple components). This section uses the idea of a *context* to define *domain* and *entity integrity constraints* for imprecise relations [3,16,17]. The context-based integrity constraints defined

in this section generalize their classical counterparts. The notion of a context will be used in later sections to formalize *imprecise FDs* and *imprecise inference compromise*, which also generalize their precise counterparts.

3.1 Contexts

A *context* C is a partition on a set $\hat{D}$ generated by a semantics-based equivalence relation ρ on $\hat{D}$. $\hat{D}$ is a subset of the underlying database domain D. Since ρ captures natural semantic equivalences between domain elements, the equivalence classes in C are sets of "closely related" (indistinguishable) elements. The set of all equivalence relations on subsets of D is denoted by $\mathcal{R}_D$; the corresponding context set is $\mathcal{C}_D$.

Contexts can be ordered by the equivalences contained in the generating equivalence relations. A "coarser" equivalence relation contains more equivalences than a "finer" equivalence relation and yields a "coarser" context with larger equivalence classes.

Definition: Let ρ and ρ' be equivalence relations in $\mathcal{R}_D$. Then, ρ' is *coarser than* ρ, i.e., $\rho' \sqsubseteq_\rho \rho$, iff $\rho \subseteq \rho'$. If C and C' are contexts induced by ρ and ρ', respectively, then C' is *coarser than* C, i.e., $C' \sqsubseteq_C C$.

An equivalence class in a finer context is a subset of an equivalence class in a coarser context. For example, the relation $\{\{a, b\}, \{c, d, e\}\} \sqsubseteq_C \{\{b\}, \{c, d\}\}$ holds. On the other hand, the contexts $\{\{a\}, \{b, c\}\}$ and $\{\{a, b\}, \{c\}\}$ are not comparable.

$(\mathcal{C}_D, \sqsubseteq_C)$ is a *complete lattice*. The coarsest context $\{D\}$ has a single (largest) equivalence class. The finest "vacuous" context is induced by the empty equivalence relation.

3.2 Domain Integrity

When enforcing a domain integrity constraint, the equivalence classes in a context act as *sieve openings* controlling the maximum imprecision of information chunks storable as tuple components. An imprecise tuple component that passes through a sieve opening in a context (i.e., it is a subset of an equivalence class) is said to be *consistent* with respect to the context. Consistent components are meaningful because equivalence classes comprise semantically-related elements. The null set is not a consistent value in this model. The maximally imprecise value from a domain D is defined as D itself.

Definition: An imprecise (set-valued) component t_i is *consistent* with respect to a context C_i iff it is a non-empty subset of an equivalence class in C_i.

Definition: An imprecise tuple $t = (t_1, t_2, ..., t_n)$ is *consistent* with respect to $(C_1, C_2, ..., C_n)$ iff each t_i is consistent with respect to context C_i.

The classical domain integrity constraint is enforced by contexts with singleton equivalence classes. Only atomic values are consistent with respect to these "precise contexts." Coarser contexts permit the storage of larger information chunks as tuple components.

3.3 Equivalence, Entity Integrity and Imprecise Relations

We now define context-based equivalence for imprecise information chunks and the related notion of entity integrity for imprecise relations. These definitions reduce to their classical counterparts for precise contexts with singleton equivalence classes.

Definition: Imprecise values (sets) t and t' are *equivalent* with respect to a context C, denoted by $t \sim_C t'$, iff t and t' are non-empty subsets of the same equivalence class in C.

A context acts as a sieve for determining the equivalence of information chunks. Since an equivalence class (sieve opening) comprises indistinguishable elements, all consistent chunks passing through the same sieve opening are considered equivalent. For precise contexts, context-based equivalence reduces to classical equality. The corresponding precise chunks are equivalent only when they are identical.

Definition: Two imprecise tuples t and t' are *redundant* with respect to contexts $C = (C_1, C_2, ..., C_n)$, i.e., $t \sim_C t'$ iff $t_i \sim_{C_i} t'_i$ for each component i.

The classical entity integrity property requires that a relation be a set of tuples. A similar property is used for imprecise databases: No two tuples in an imprecise relation may be "identical." In this case, however, "identical" is defined as equivalence with respect to contexts, and a "set" may not contain multiple equivalent objects.

Definition: An *imprecise relation scheme* $R_{(A,C)}$ is a collection of attributes $A = (A_1, A_2, ..., A_n)$ with associated contexts $C = (C_1, C_2, ..., C_n)$.

Definition: An *imprecise database relation* r with underlying scheme $R_{(A,C)}$ is a set of non-redundant tuples with respect to the contexts in C.

An imprecise relation scheme is a collection of attributes and associated contexts. Since a classical relation scheme has precise contexts on all its attributes, a classical database relation can only hold precise information. An imprecise relation scheme has coarser contexts on some or all of its attributes. This enables the corresponding imprecise relation to consistently hold imprecise information chunks.

Entity integrity is preserved by subsuming redundant tuples. In a classical relation, each set of identical tuples is simply replaced by one of the tuples. However, extra consideration must be given to an imprecise relation because it is possible to have redundant imprecise tuples which are different from each other. Entity integrity is maintained in an imprecise (or precise) relation by "merging" a block of redundant tuples into a single non-redundant tuple.

Definition: The *merge* of two tuples t and t' is $u = (u_1, u_2, ..., u_n)$ where $u_i = t_i \cup t'_i$.

The merge operation has an important role in imprecise inference analysis, especially for generating imprecise relations to satisfy induced FDs (recall the example in the previous section). Imprecise inference analysis is discussed in detail in Section 4.

Name	*Experience*	*Salary*
{*John*}	[7, 10)	{65*K*}
{*Bill*}	{2}	[25*K*, 40*K*)

An example imprecise relation is presented above. It is defined with respect to a precise context on the *Name* attribute, and coarser contexts {[0, 5), [5, 10)} and {[0*K*, 50*K*), [50*K*, 100*K*)}, on the *Experience* and *Salary* attributes, respectively. The tuples in the relation are consistent and non-redundant with respect to these contexts. However, the tuples become redundant if the coarser contexts, {*Men's Names*}, {[0, 10)}, and {[0*K*, 100*K*)}, are used for the *Name*, *Experience* and *Salary* attributes, respectively. The new merged relation contains a single tuple obtained by taking the set union of corresponding components in the redundant tuples. Note that precise information is correctly expressed using singleton sets. However, for simplicity we will represent precise information as atomic values in the following sections.

4 Imprecise FDs and Imprecise Inference Channels

This section introduces the main concepts necessary for analyzing imprecise inference. It clarifies the notion of an imprecise FD and presents inference axioms for determining closure. Two of these axioms correspond to abduction and partial deduction. Finally, this section defines the notion of an imprecise inference channel. Since the definitions generalize their classical counterparts, the imprecise inference methodology also can be used for analyzing precise inference.

The existence of an imprecise FD implies that if some tuple components satisfy certain equivalences, then other tuple components must exist and their values must be equivalent [7,17]. This notion extends the equality-based classical FD to one based on equivalence with respect to contexts. Imprecise FDs can specify constraints on precise and imprecise data. Examples of imprecise FDs are: *Engineers have starting salaries of about 40K*, and *Approximately equal qualifications and more or less equal experience demand similar salary.* Clearly, such FDs have a key role in imprecise inference analysis.

Definition: An *imprecise FD:* $X(C_X) \rightarrow Y(C_Y)$ holds in scheme $R_{(A,C)}$ iff for all tuples t, t' in every extension r of $R_{(A,C)}$, $t \sim_{C_X} t'$ implies $t \sim_{C_Y} t'$.

Lemma 4.1: The imprecise FD: $X(C_X) \rightarrow Y(C_Y)$ is a classical FD when C_X and C_Y are precise contexts.

Determining the closure of inference for a set of imprecise FDs is critical to analyzing imprecise inference. The classical model uses Armstrong's axioms in defining the FD-based inference closure. Counterparts to Armstrong's axioms exist for context-based imprecise FDs [17].

Lemma 4.2: Imprecise FDs satisfy Armstrong's Axioms:

$Y(C_Y) \subseteq X(C_X) \subseteq U(C_U)$ implies $X(C_X) \rightarrow Y(C_Y)$ (reflexivity)

$X(C_X) \rightarrow Y(C_Y)$ implies $XZ(C_X C_Z) \rightarrow YZ(C_Y C_Z)$ where $Z(C_Z) \subseteq U(C_U)$ (augmentation)

$X(C_X) \rightarrow Y(C_Y)$ and $Y(C_Y) \rightarrow Z(C_Z)$ imply $X(C_X) \rightarrow Z(C_Z)$ (transitivity).

In addition to the counterparts to Armstrong's axioms, new axioms specific to imprecise FDs exist [21]. The following inference axiom states that if an information chunk x determines some chunk y using an imprecise FD, then information more precise than x can determine information less precise than y.

Lemma 4.3: $X(C'_X) \rightarrow Y(C_Y)$ implies $X(C_X) \rightarrow Y(C'_Y)$ for all $C'_X \sqsubseteq_C C_X$, $C'_Y \sqsubseteq_C C_Y$.

As seen in Section 2, deductive and abductive inference can be unified using the idea of an induced FD. In the context model, an induced FD is an imprecise FD defined in terms of "induced contexts" [21]. For a relation containing attributes X and Y, the induced context is the finest context for Y that yields an imprecise FD from X to Y with context C_X for X. The induced context is computed as the greatest lower bound (*glb*) of equivalence classes merged in Y according to values in X. It represents the greatest lower bound on the inferred information and always exists because the set of contexts is a complete lattice.

Definition: Let F be the set function induced by the mapping $X \rightarrow Y$ from X tuple components to Y components and let $\langle t[X] \rangle_{C_X}$ be the set of tuple components that are equivalent to $t[X]$ with respect to C_X. An *induced context* on Y, denoted by $\mathcal{I}_{X \rightarrow Y}(C_X)$, is constructed by making the set of Y tuple components in $F(\langle t[X] \rangle_{C_X})$ redundant for each $t[X]$ and then taking the greatest lower bound (*glb*) of this collection of contexts.

Having clarified the idea of an induced context, we now present the inference axioms for abduction and partial deduction (Lemmas 4.4 and 4.5, respectively). $\perp_Y$ in Lemma 4.5 denotes the coarsest context for Y. It is required because information about Y is not used in partial deduction.

Lemma 4.4: $X(C_X) \rightarrow Y(C_Y)$ implies $Y(C_Y) \rightarrow X(\mathcal{I}_{Y \rightarrow X}(C_Y))$ and for any C'_X such that $Y(C_Y) \rightarrow X(C'_X)$ holds, $C'_X \sqsubseteq_C \mathcal{I}_{Y \rightarrow X}(C_Y)$.

Lemma 4.5: $XY(C_X C_Y) \rightarrow Z(C_Z)$ implies $X(C_X) \rightarrow Z(\mathcal{I}_{XY \rightarrow Z}(C_X \perp_Y))$.

Precise inference channels are composed of a "use set" of FDs [12]. Likewise, imprecise inference channels are composed of a use set of imprecise FDs. An imprecise inference channel is a chain of imprecise FDs, usually represented as a directed acyclic graph (dag).

Definition: An *imprecise inference channel* from $X(C_X)$ to $Z(C_Z)$ is a sequence of imprecise FDs, $\mathcal{F} = F_1, F_2, \ldots, F_n$, such that $X(C_X) = LHS(F_1)$, $Z(C_Z) = RHS(F_n)$, and $RHS(F_i) = LHS(F_{i+1})$.

The notion of an imprecise inference channel and its role in inference analysis and control are clarified using a series of examples in the following section.

5 Imprecise Inference Control Methodology

This section outlines the basic imprecise inference control methodology. Imprecise inference analysis and its application in a practical inference control methodology are described in detail.

5.1 Analyzing Imprecise Inference

The definitions in the previous section provide the formal mechanism necessary for the rigorous analysis of imprecise inference. Imprecise inference analysis primarily involves the determination of the inference closure. This closure is computed using the inference axioms defined in the previous section. The inference closure can be visualized by materializing imprecise relations which correspond to the FDs induced within relations and across relations with semantically equivalent attributes. The imprecise relations then are composed to create new relations corresponding to potentially compromising inference channels.

Hypergraphs are used to represent FD-based inference channels. The formal definition of a hypergraph used in our work is given below.

Definition: A *hypergraph* is a collection of edges E_H and vertices $V_H = 2^S$ on some basis set S. A vertex is a subset of S and an edge is a pair of vertices.

Given a relation and a set of FDs (preferably a minimum cover), it is a simple task to derive a hypergraph whose edges connect vertices containing sets of attributes. Such a hypergraph facilitates exploration of the inference closure for the database being analyzed. We present a simple algorithm for constructing a hypergraph from a global relation scheme and a set of FDs.

Hypergraph Construction Algorithm

G: Global scheme; R: Relation scheme; F: Set of FDs; A: Relational attribute
$\forall R \in G, \forall A \in R$: Instantiate a node labelled A_R.
$\forall$ FD: $X \rightarrow Y \in F$:
Create a hypernode referencing all nodes with attributes in X if no such hypernode exists.
Create a hypernode referencing all nodes with attributes in Y if no such hypernode exists.
Add a hyperedge between these two hypernodes.

Note that the hyperedges are bidirectional; thus, they capture abductive as well as deductive paths. The constructed hypergraph expresses all inference paths in the database system. It must be analyzed to detect all potentially compromising inference channels.

The following simple example illustrates imprecise inference analysis and clarifies many of the previously-defined concepts, including context-based imprecise relation, imprecise FD and imprecise inference channel. Consider a database containing the three relations, $R_1(Emp_1, Sal_1)$, $R_2(Sal_2, Tax_2)$ and $R_3(Emp_3, Tax_3)$. Note that R_1 and R_2 are imprecise relations while R_3 is a precise relation.

$R_1(Emp_1, Sal_1)$

Emp_1	Sal_1
$John$	$[0K, 50K)$
$Mary$	$[50K, 100K)$
Joe	$[0K, 50K)$
$Jane$	$[0K, 50K)$

$R_2(Sal_2, Tax_2)$

Sal_2	Tax_2
$[0K, 10K)$	10%
$[10K, 20K)$	15%
$[20K, 35K)$	18%
$[35K, 50K)$	25%
$[50K, 80K)$	30%
$[80K, 100K)$	35%

$R_3(Emp_3, Tax_3)$

Emp_3	Tax_3
$John$	18%
$Mary$	30%
Joe	25%
$Jane$	18%

Relation R_1 expresses "common knowledge" possessed by low users that salaries are in the $[0K, 50K)$ or $[50K, 100K)$ range. This relation corresponds to "catalytic data" [8] added to a database by the DBA to materialize imprecise inference channels due to common knowledge. The coarse context $C_{Sal_1} = \{[0K, 50K), [50K, 100K)\}$ enables imprecise salaries to be stored consistently in R_1. A precise context is used for attribute Emp_1. The imprecise FD: $Emp_1 \rightarrow Sal_1$ holds in R_1 for contexts C_{Emp_1} and C_{Sal_1}. (A precise FD would have held between the same attributes had the relation R_1 been precise.) Since this example deals with the inference of salary values, let the inference channel $IC_1 = \{Emp_1 \rightarrow Sal_1\}$. IC_1 is a use set containing a single imprecise FD.

Relation R_2 is a tax table. This table could already exist in the database or it could be added to the database as a catalytic relation conveying common knowledge. The table contains imprecise information in that multiple salary values map to a single tax rate; it could be expressed less concisely by a table containing only precise values. The context $C_{Sal_2} = \{[0K, 10K), [10K, 20K) [20K, 35K), [35K, 50K), [50K, 80K), [80K, 100K)\}$ is used for the salary attribute. The context used for Tax_2 is precise. The imprecise FD: $Sal_2 \rightarrow Tax_2$ holds in R_2 for contexts C_{Sal_2} and C_{Tax_2}.

Relation R_3 contains precise information and uses precise contexts for all its attributes. The precise FD: $Emp_3 \rightarrow Tax_3$ holds in R_3.

The inference axioms produce the channel IC_2: $\{Emp_1 \rightarrow Emp_3, Emp_3 \rightarrow Tax_3, Tax_3 \rightarrow Tax_2, Tax_2 \rightarrow Sal_2, Sal_2 \rightarrow Sal_1\}$. Notice that some FDs in the use set arise from the semantic equivalence of attributes, e.g., FD: $Emp_1 \rightarrow Emp_3$. The salary information inferred through inference channel IC_2 is presented in Relation R_{IC_2} below. The new relation is obtained by joining relations R_3 and R_2 as specified in the use set for IC_2.

$R_{IC_2}(Emp, Sal)$

Emp	Sal
$John$	$[20K, 35K)$
$Mary$	$[50K, 80K)$
Joe	$[35K, 50K)$
$Jane$	$[20K, 35K)$

We can see that a potential imprecise inference compromise exists because the derived relation R_{IC_2} has salary

information of finer granularity than the original relation R_1. More precisely, the compromise exists because the derived inference channel IC_2 is not coarser than the original channel IC_1. The induced context C_{Sal} in R_{IC_2} (and IC_2) is equal to $\{[20K, 35K), [35K, 50K), [50K, 80K)\}$. It is not coarser than $C_{Sal_1} = \{[0K, 50K), [50K, 100K)\}$ in R_1 (and IC_1).

5.2 Controlling Imprecise Inference

To control and ultimately eliminate compromising imprecise inferences, it is necessary to specify a set of imprecise inference channels considered to be secure. This *compromise specification set* is defined by the DBA. Suspect channels in a database are compared with these secure channels. A potential security compromise exists when a suspect channel allows the inference of information which is not coarser than information inferred by any secure channel with a finer RHS context. Potentially compromising imprecise inference channels are eliminated by hiding relational attributes or by "clouding" data manifested by imprecise FDs in the compromising channels.

The following algorithm describes the basic inference control methodology. Note that the compromise specification set is to be defined by the DBA. The definitions of *compromise*() and *eliminate*(), which detect and eliminate potentially compromising inference channels, respectively, will be provided later.

Inference Control Algorithm

Define compromise specification set.
Construct hypergraph from global schema G and FD set F.
P: Set of paths in hypergraph
For each p_i in P: If *compromise*(p_i), then *eliminate*(p_i).

5.2.1 Compromise Specification and Detection

In a typical scenario, the DBA first identifies the set of imprecise inference channels called the *compromise specification set*. This set conveys the finest information that can be inferred without compromising database security. It typically contains simple channels composed of FDs holding in the database relations being analyzed and those holding in additional catalytic relations. As mentioned earlier, catalytic relations [8] are additional relations added to the database being analyzed. They convey common knowledge about secure attribute values and are used to materialize potentially compromising imprecise inferences that might otherwise go unnoticed. A suitable assumption used when constructing the compromise specification set is that imprecise FDs and inference channels within a relation are secure. Compromising inferences across relations can occur when attributes in different relations are semantically equivalent. A similar assumption is used in the DISSECT system.

We examine individual inference paths in the hypergraph constructed from inference analysis for suspected compromise. For inference path exploration, the hypergraph can be reduced to a regular graph whose vertices are hyperdges [21]. That is, a hyperedge $E_{(a,b)}$ with hypernodes a and b becomes a vertex $V_{E_{(a,b)}}$ in a regular graph. Two such vertices, V_{E_1} and V_{E_2}, are connected if the intersection of a hypernode of attributes from V_{E_1} and a hypernode of attributes from V_{E_2} is nonempty. Finding inference channels to test for potential compromise now reduces to path enumeration for regular graphs. The algorithm for hypergraph reduction is presented below.

Hypergraph Reduction Algorithm

E_H: Set of hypergraph edges; V_H: Set of hypergraph vertices
E_R: Set of regular graph edges; V_R: Set of regular graph vertices
For each $E_{(a,b)} \in E_H$: Add $V_{E_{(a,b)}}$ to V_R.
For each $V_{E_{(a,b)}}$, $V_{E_{(c,d)}}$ in V_R:
If $\{a \cup b\} \cap \{c \cup d\} \neq \emptyset$ then add $E_{(V_{E_{(a,b)}}, V_{E_{(c,d)}})}$ to E_R.

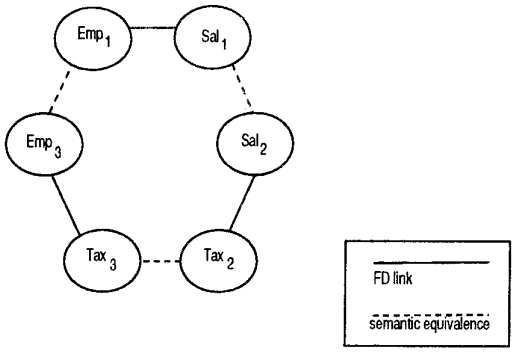


Figure 1: Example Hypergraph

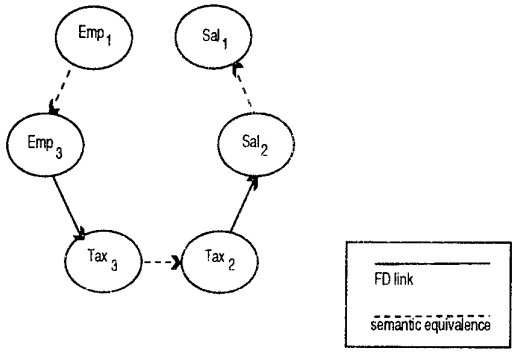


Figure 2: Inference Channel: IC2

A suspect inference channel originating at X that allows the inference of information about Y can be deemed secure when that information is coarser than the inferred information of any inference channel from $X \rightarrow Y$ in the compromise specification set. On the other hand, a potential security compromise exists, i.e., $compromise(p_i)$ is true, when there does not exist a secure inference channel in the compromise specification set such that the information inferred from the suspect channel is coarser than the information inferred from the secure channel. We use the context model to formalize the notion of a potentially compromising imprecise inference channel that is used in our inference control algorithm.

Definition: A derived inference channel IC' for $X(C_X) \rightarrow Y(C'_Y)$ is a *potentially compromising imprecise inference channel*, i.e., $compromise(IC')$ is true, iff C'_Y is not coarser than C_Y, i.e., $C'_Y \sqsubseteq_C C_Y$ does not hold, for some IC for $X(C_X) \rightarrow Y(C_Y)$ in the compromise specification set.

For the example in the previous subsection, the compromise specification set is equal to $\{Emp_1 \rightarrow Sal_1, Sal_2 \rightarrow Tax_2, Emp_3 \rightarrow Tax_3\}$. The inferences possible in the example database are represented in the hypergraph in Figure 1. A potentially compromising inference channel exists because the derived channel IC_2: $\{Emp_1 \rightarrow Emp_3, Emp_3 \rightarrow Tax_3, Tax_3 \rightarrow Tax_2, Tax_2 \rightarrow Sal_2, Sal_2 \rightarrow Sal_1\}$ has a context for C_{Sal_1} which is not coarser than the corresponding context for inference channel IC_1: $\{Emp_1 \rightarrow Sal_1\}$ in the compromise specification set. The derived inference channel IC_2 is presented in Figure 2.

Detecting imprecise inference compromises is an NP-complete problem [21]. This follows from the fact that imprecise inference compromise detection generalizes the problem of detecting precise FD-based inference channels; the latter was shown to be NP-complete by Su and Ozsoyoglu [19,20]. Exhaustive search is the obvious strategy for detecting compromising channels. Large databases will require the use of heuristics for compromise

detection. Effective and practical heuristics for compromise detection are presented in [21]. Allowing for user interaction during the detection phase enhances the quality of the search.

5.2.2 Compromise Elimination

Securing a database system from imprecise inference attacks based on FDs mandates securing all potentially compromising imprecise inference channels. Each channel can be secured by hiding (upgrading) relational attributes [18-20]. Alternatively, the compromising inference channels may be secured by appropriately "clouding" tuple components [16,17,21]. Information used in a potentially compromising inference channel is clouded so that the inferred information is coarser than that inferred using any related secure inference channel. When a suspect channel has no secure counterpart in the compromise specification set, it must be secured according to policies specified by the DBA.

In our example it is necessary only to secure the inference channel IC_2. To illustrate the technique, we secure the database using information clouding. This is accomplished by applying the finest clouding of the channel to eliminate the compromise. Selecting the channels to cloud and determining how much to cloud is an NP-complete problem, but for short inference channels a simple heuristic suffices [7,21]. According to this heuristic, the inference channel is traced backwards and information in the first acceptable attribute is clouded. An attribute in an imprecise FD is not acceptable for clouding if the mapping is common knowledge (e.g., FD: $Sal_2 \rightarrow Tax_2$ represents the tax table), or if the dependency arises due to the semantic equivalence of attributes (e.g., FD: $Emp_1 \rightarrow Emp_3$). The clouding algorithm presented below effectively secures compromising inference channels. It is an appropriate specification of the function *eliminate*() in the inference control algorithm.

Clouding Algorithm

$\mathcal{X} = (X_1, \ldots, X_n)$: Compromising inference channel
$\mathcal{C}_{X_i}$: Context for attribute X_i
Choose $\mathcal{Y} = (Y_1, \ldots, Y_m)$ to be a related secure channel
(where X_n and Y_m refer to the same attibute).
Set $i = n$.
while cannot cloud X_i, $i = i - 1$
Set $C(X_i) = \mathcal{I}_{\mathcal{Z}}(\mathcal{I}_{\mathcal{Y}})$ (where $\mathcal{Z} = (X_n, \ldots, X_i)$).

Applying this heuristic to our example, we select attribute Tax_3 in the FD: $Emp_3 \rightarrow Tax_3$ for clouding (or hiding). We cloud it by determining what can be inferred about Tax_3 using the *inverse channel*, IC_2^{-1}, which infers Emp_1 from Sal_1 using the imprecise values known for Sal_1, i.e., $[0K, 50K)$, $[50K, 100K)$. This inference channel yields imprecise tax values of {10%, 15%, 18%, 25%} and {30%, 35%} from imprecise salary values of $[0K, 50K)$ and $[50K, 100K)$, respectively. The relation $R_3(Emp_3, Tax_3)$ is coarsened appropriately to yield the clouded relation R_3^* below:

$R_3^*(Emp_3, Tax_3)$

Emp_3	Tax_3
John	{10%, 15%, 18%, 25%}
Mary	{30%, 35%}
Joe	{10%, 15%, 18%, 25%}
Jane	{10%, 15%, 18%, 25%}

The final database contains relation $R_1(Emp_1, Sal_1)$ representing *a priori* knowledge, the tax table $R_2(Sal_2, Tax_2)$, and $R_3^*(Emp_3, Tax_3)$ in which Tax_3 values are clouded to eliminate the imprecise inference channel. Note

that in the new database the inference channel IC_2 yields information which is coarser than that provided by the secure channel IC_1. This verifies that the channel is secure. The secured database is shown below.

$R_1(Emp_1, Sal_1)$

Emp_1	Sal_1
$John$	$[0K, 50K)$
$Mary$	$[50K, 100K)$
Joe	$[0K, 50K)$
$Jane$	$[0K, 50K)$

$R_2(Sal_2, Tax_2)$

Sal_2	Tax_2
$[0K, 10K)$	10%
$[10K, 20K)$	15%
$[20K, 35K)$	18%
$[35K, 50K)$	25%
$[50K, 80K)$	30%
$[80K, 100K)$	35%

$R_3^*(Emp_3, Tax_3)$

Emp_3	Tax_3
$John$	{10%, 15%, 18%, 25%}
$Mary$	{30%, 35%}
Joe	{10%, 15%, 18%, 25%}
$Jane$	{10%, 15%, 18%, 25%}

6 Conclusions

Research on imprecise inference by Morgenstern [13,14] and Garvey and Lunt [4,6] has highlighted the need to develop formal, yet practical, techniques for dealing with the difficult problem of imprecise inference control. The context formalism presented in this paper meets these requirements. Contexts provide a powerful and systematic approach for modeling imprecise inference. They also define integrity constraints for imprecise relations and the key notion of an imprecise FD which generalizes precise deduction and unifies the primitive imprecise inference mechanisms of abduction and partial deduction. Whereas other imprecise (partial) inference models estimate the probability or possibility of making inferences, the context-based methodology examines the actual imprecise values inferred in a database extension. Imprecise inference analysis is performed by transforming a precise database, possibly augmented with additional catalytic relations conveying *a priori* knowledge available to low users, into an equivalent imprecise database. This technique, which is similar to those used in deductive databases, effectively models the propagation of compromising imprecise inferences in MLS databases. With minimal modifications, the same inference analysis technique can be applied to knowledge discovery or database mining.

The inference control formalism described in this paper provides a foundation for the development of practical inference controllers for database systems. The DISSECT prototype developed at SRI International [5,15,18] demonstrates the utility of an interactive tool for controlling database inference. The context-based model is particularly suited for implementation in an interactive environment. The flexibility of the model will enable the resulting tools to secure classical, MLS, and imprecise databases from precise and imprecise inference attacks.

7 References

[1] L.J. Buczkowski, Database inference controller, in *Database Security III: Status and Prospects*, D.L. Spooner and C. Landwehr (Eds.). Elsevier Science, New York, pp. 311-322, 1990.

[2] D.E. Denning, Commutative filters for reducing inference threats in multilevel database systems, *Proceedings of the 1985 IEEE Symposium on Research in Security and Privacy*, pp. 134-146, 1985.

[3] S. Finnerty and S. Shenoi, Abstraction-based query languages for relational databases, in *Advances in Fuzzy Theory and Technology, Volume 1*, P.P. Wang (Ed.), Bookwrights, Durham, North Carolina, pp. 195-218, 1993.

[4] T.D. Garvey and T.F. Lunt, Cover stories for database security, *Proceedings of the Fifth IFIP WG11.3 Workshop on Database Security*, November 1991.

[5] T.D. Garvey, T.F. Lunt, X. Qian and M.E. Stickel, Toward a tool to detect and eliminate inference problems in

the design of multilevel databases, *Proceedings of the IFIP TC11/WG11.3 Sixth Working Conference on Database Security*, pp. 159-177, 1992.

[6] T.D. Garvey, T.F. Lunt and M.E. Stickel, Abductive and approximate reasoning models for characterizing inference channels, *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, June 1991.

[7] J. Hale, S. Finnerty and S. Shenoi, Analyzing inference in fuzzy database systems, submitted to the *Third IEEE International Conference on Fuzzy Systems*, Orlando, Florida, June 29 - July 1, 1994.

[8] T.H. Hinke, Inference aggregation detection in database management systems, *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pp. 96-106, 1988.

[9] T.H. Hinke and H. Delugach, Aerie: An inference modeling and detection approach for databases, *Proceedings of the IFIP TC11/WG11.3 Sixth Working Conference on Database Security*, 1992.

[10] T.F. Lunt, Aggregation and inference: Facts and fallacies, *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, pp. 102-109, 1985.

[11] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman and W.R. Shockley, The SeaView security model, *IEEE Transactions on Software Engineering*, vol. 16, pp. 593-607, 1991.

[12] D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland, 1983.

[13] M. Morgenstern, Security and inference in multilevel database and knowledge base systems, *Proceedings of the ACM International Conference on the Management of Data (SIGMOD)*, pp. 357-373, 1987.

[14] M. Morgenstern, Controlling logical inference in multilevel database systems, *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pp. 245-255, 1988.

[15] X. Qian, M.E. Stickel, P.D. Karp, T.F. Lunt and T.D. Garvey, Detection and elimination of inference channels in multilevel relational database systems, *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, pp. 196-205, 1993.

[16] S. Shenoi, Multilevel database security using information clouding, *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, pp. 483-488, 1993.

[17] S. Shenoi, Fuzzy Sets, information clouding and database security, to appear in *Fuzzy Sets and Possibility Theory in Database Management Systems*, P. Bosc and J. Kacprzyk (Eds.), Omnitech Press, Warsaw, Poland, and Physica-Verlag, Heidelberg, Germany, 1994.

[18] M.E. Stickel, X. Qian, T.F. Lunt and T.D. Garvey, Inference channel detection and elimination, Elin A003: Second Interim Report, Computer Science Laboratory, SRI International, Menlo Park, California, 1993.

[19] T.A. Su and G. Ozsoyoglu, Data dependencies and inference control in multilevel relational database systems, *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*, pp. 202-211, 1987.

[20] T.A. Su and G. Ozsoyoglu, Controlling FD and MVD inferences in multilevel relational database systems, *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, pp. 474-485, 1991.

[21] J. Threet, *Designing Secure Relational Databases*, M.S. Thesis, Computer Science, University of Tulsa, Tulsa, Oklahoma, 1993.

[22] J. Threet and S. Shenoi, Controlling deduction and abduction in clouded relational databases, presented at the *1993 IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 24-26, 1993.

[23] M.B. Thuraisingham, Security checking in relational database management systems augmented with inference engines, *Computers and Security*, vol. 6, pp. 325-333, 1987.

[24] M.B. Thuraisingham, The use of conceptual structures for handling the inference problem, Technical Report M90-55, The MITRE Corporation, Bedford, Massachusetts, 1990.

Hypersemantic Data Modeling for Inference Analysis

Donald G. Marks and Leonard J. Binns
Office of INFOSEC Computer Science
Department of Defense
Ft. Meade, MD

Bhavani M. Thuraisingham
The MITRE Corporation
Bedford, MA

ABSTRACT

This paper describes a hypersemantic data model for representing multilevel database applications. This data model, which is called multilevel knowledge data model (MKDM), incorporates constructs from both data models and knowledge models. An associated data definition language called multilevel knowledge data language (MKDL) and a graphical representation scheme are also described. Finally, knowledge transformation as well as inference analysis issues are discussed. Our goal is to develop a uniform representation and specification scheme which can not only be used for inference analysis, but which can also be transformed into other representation schemes so that existing inference analysis tools can be applied.

1. INTRODUCTION

It is possible for users of any database management system to draw inferences from the information that they obtain from the databases. The inferred knowledge could depend only on the data obtained from the database system or it could depend on some prior knowledge possessed by the user in addition to the data obtained from the database system. The inference process can be harmful if the inferred knowledge is something that the user is not authorized to acquire. That is, a user acquiring information which he is not authorized to know has come to be known as the inference problem in database security.

We are particularly interested in the inference problem which occurs in a multilevel operating environment. In such an environment, the users are cleared at different security levels and they access a multilevel database where the data is classified at different sensitivity levels. A multilevel secure database management system (MLS/DBMS) manages a multilevel database where its users cannot access data to which they are not authorized. However, providing a solution to the inference problem, where users issue multiple requests and consequently infer unauthorized knowledge, is beyond the capability of currently available MLS/DBMSs.

Morgenstern was the one of the first to investigate the inference problem for MLS/DBMSs [MORG87]. Since then, several efforts have been reported. One of the major approaches to handling the inference problem is to design the multilevel database in such a way that certain security violations are prevented (see for example the work of Binns [BINN92], Burns [BURN92], Hinke et al. [HINK92], Garvey et al. [GARV92], Smith [SMIT90], and Thuraisingham [THUR90]). That is, the security constraints, which are rules that assign security levels to the data, are processed during multilevel database design and subsequently the schemas are assigned appropriate security levels. While some of the proposed solutions focus on representing the multilevel database application using conceptual structures developed for knowledge-based system applications and subsequently reasoning about the application using deduction techniques (see for example [HINK92, GARV92, and THUR90]), some focus on developing tools which generate new relational database schemas given the original relational database schemas and the security constraints (see for example [BINNS92]), and some others are proposing the use of semantic data models developed for database design to design the multilevel database also (see for example [BURN88]).

While the three approaches complement each other, each of them uses different representation schemes and different inference analysis tools. That is, the approaches are highly specialized and therefore, if we use one tool for inference analysis, then we cannot take advantage of the useful features offered by the other tools. Since no tool can handle all types of inference problems and the problems handled by each tool are not the same, some of the inference problems cannot be handled with the current approaches. What we need is a powerful uniform representation scheme which can be transformed into specific representation schemes without much complexity.

Once this is done, the various inference analysis tools can be applied. This way one can take advantage of the tools that have been developed without having to re-invent the wheel. The purpose of this paper is to describe the essential points of this uniform representation method. That is, we are not interested in simply developing yet another model for representing the application and conducting inference analysis. Our goal is to develop a uniform representation and specification scheme which can not only be used for inference analysis, but which can also be transformed into other representation schemes so that existing inference analysis tools can be applied.

The data model that is to be used to develop the uniform representation scheme should capture not only the entities of the application and the relationships between them, but also provide the means to specify constraints, heuristics and other complex relationships. What is needed is a model for bridging the gap between data and knowledge base systems. This is because data models have traditionally focused on representing data while knowledge models have focused on knowledge representation for expert systems. An integrated model would be appropriate for capturing the structural properties, the semantics, and the constraints of the application. Such integrated models have been called hypersemantic data models in the literature [POTT89]. This paper discusses the use of hypersemantic data modeling for capturing the semantics as well as the structural aspects of multilevel database applications, describes the generation of multilevel database schemas from this representation, and shows how inference analysis tools could be applied.

The hypersemantic data model that we have developed is called Multilevel Knowledge Data Model (MKDM). This model extends Potter et al.'s [POTT89] hypersemantic data model for multilevel database applications. It integrates data and knowledge and captures the static as well as temporal aspects of the application. We have also developed a graphical representation scheme called GRAPHICAL-MKDM as well as a specification language called Multilevel Knowledge Data Language (MKDL) for MKDM. The representation schemes that we have developed are general enough to be transformed into conceptual structures such as semantic nets, logic programming languages, as well as SQL. This way, the inference analysis tools that have been developed for other representation schemes could be applied. It should also be noted that inference analysis tools could be developed for specifications in GRAPHICAL-MKDM and MKDL. Such tools could find those potential inference problems that can be uncovered with the complex constructs of MKDM and which are not straightforward with the other representation schemes.

The organization of this paper is as follows. The hypersemantic data model that we have developed as well as the associated specification language are discussed in Section 2. In particular, the essential constructs of MKDM, MKDL (the language for specifying MKDM), and a graphical representation scheme are described. In Section 3, we describe how the graphical representation scheme and MKDL specification can be transformed into some of the representation schemes proposed by others in the literature. In particular, we show how MKDM-based representation of an application can be transformed into representations based on conceptual structures, logic programming specification, and extended SQL specification. The types of inferences to be handled as well as applying different inference analysis tools on the various representation schemes of the application are described in Section 4. Related work is discussed in Section 5. The paper is concluded in Section 6 with a discussion of future considerations.

2. DETAILS OF THE HYPERSEMANTIC DATA MODEL

This section describes the details of the hypersemantic data model that we have developed. In section 2.1 we describe MKDM. In section 2.2, we describe MKDL. Our graphical representation scheme is described in section 2.3.

2.1 MULTILEVEL KNOWLEDGE DATA MODEL

2.1.1 OVERVIEW

In this section, we describe the model that we have developed for representing multilevel database applications. This model, which incorporates constructs from semantic data models and knowledge models, is called a multilevel knowledge data model (MKDM). It incorporates data, knowledge, and security semantics of an application. As in most semantic models, we use the notion of an object to represent any structural entity in the application. Such an entity could be a person Joe, or a dog Lassie, or an airplane AAA. Each entity consists of a set of properties which describe that entity. The essential constructs of the model are the following:

(i) Classification: objects with similar properties (which are also called attributes) are grouped into an object-type via the "instance-of" relationship.

(ii) Generalization: a subset of the objects of an object-type which have common properties are grouped into another object-type via the "is-a" relationship.

(iii) Aggregation: an object is composed into multiple components via the "is-part-of" relationship.

(iv) Membership: a collection of object-types are abstracted into a new object type via the "is-a-member-of" relationship. An instance of this new type will consist of a collection of objects, one from each object type which formed the new type.

(v) General constraints: a restriction is placed on some aspect of an object (such as the value of its property), operation, or relationship via the "is-constraint-on" relationship.

(vi) Heuristics: information derivation mechanism is attached via the "is-heuristics-on" relationship.[1]

(vii) Temporal: specific object types are related by synchronous or asynchronous relationships.

(viii) Security constraints: a restriction is placed on the security level of an object, an object-type, an attribute, or on the association between the property of an object and the value of this property. The assignment of the security level may depend on the content, context, or time. Each classification constraint may have a corresponding explanation for the restriction.

The following example illustrates briefly the essential points in the modeling constructs. An example of the classification construct is the grouping of all students into a STUDENT type. An example of the generalization construct is grouping all students over age 25 into a type called ADULT-STUDENT. An example of an aggregation construct is a book MATH-A which consists of the components INTRODUCTION, CALCULUS, ALGEBRA, GEOMETRY, and CONCLUSION. An example of membership construct is (STUDENT, TEACHER) whose instances are the (student, teacher) pairs. An example of a general constraint is a rule that each student is bounded by the number of courses taken depending on his year. An example of a heuristic construct is the inference rule that if a student is in his senior year, then he must have taken at least 20 courses. An example of a temporal construct is that before a student starts his thesis he must finish his qualifying exams and his oral exams. An example of a security constraint is assigning the Secret level to the association between the GRADE property of the STUDENT object-type and its value.

In sections 2.1.2. to 2.1.9 we will discuss the details of each construct with examples. It should be noted that security constraints are a special type of construct. They are used in the inference analysis process during the design of the application. They are also used by the inference analysis tools to generate the database schema.

2.1.2 CLASSIFICATION CONSTRUCT

The purpose of MKDM is to define the objects, its properties, and the associated security levels. Each object has associated with it a security level which we assume is the existence level of the object. That is, if an object's level is L, then its existence is known at level L or higher.

The classification construct is used to group objects with similar properties into an object-type. Now, the question is, should an object-type have a security level? Since objects have levels, it is reasonable to assume that an object-type which groups objects also has a level. This level is the existence level of the object-type. The relationship between the levels of the object and its object- type should be such that it is not possible to have a security violation via inference. For example, if an object's level is L1, its object-type's level is L2, and $L1 \leq L2$, then one could infer at L1 information about the object-type. So, it would be safer to have $L1 \geq L2$.

---

[1]Integrity constraints are a form of general constraints. General constraints specify causal relationships while heuristics do not. An example of a general constraint is "if a student's GPA is less than 2.0, then he cannot take Math 231". There is a cause for a student not to take Math 231. An example of a heuristic constraint is " if a student is in his Senior year, then he must have taken 20 courses. Here, the fact that a student has taken 20 courses has nothing to do with him being in his senior year.

A model should also capture the properties of objects which are specified in the object-type classifying that object. Now, what should be the relationship between the level of an object-type and the level of its attributes? What is the connection between the level of an attribute and the level of its values?[2] It is felt that at higher levels one could have information about additional attributes for a class. Also, if the level L1 of the attribute is dominated by the level L2 of the object-type, then at level L1, one could infer the existence of the object-type. Therefore, it is safer to have the property that L2 ≤ L1. Now, knowing that A is an attribute of object-type O does not mean that one can read the value of the attribute. So, the level of the value (which is actually the level of the association between the attribute and its value) dominates the level of the attribute.[3] Otherwise, one could read the value and infer that there is an attribute. The essential points are illustrated in figure 1. STUDENT class is Unclassified with instances which are Unclassified and Secret. The attributes Name and GPA are Unclassified while the attribute Bank-account is Secret. But the value of the GPA attribute (i.e., the association between the GPA attribute and its value) is Secret. Figure 1 also illustrates an Unclassified instance of the object-type STUDENT. This instance has name Jane, GPA of 3.8 and bank account ID of ppp. The association between the name attribute and its value is Unclassified while the associations between the other two attributes and their respective values are Secret.[4]

Other complex types of security constraints should also be taken into consideration during the inference analysis process. For example, one could classify the GPA value at the Secret level only if the student is attending a military academy. Another constraint would be to classify the GPA value at the Secret level and enforce a general constraint that the teacher attribute of a student implies the GPA value. A third constraint would be to classify the association between the value of the GPA attribute and the value of the name attribute at the TopSecret level. Such constraints should be taken into consideration when the security levels are assigned. For example, if the teacher implies the GPA (assuming that certain teachers only teach students with a higher GPA value), if GPA values are Secret, then so should the teacher values. Otherwise, by inference, an Unclassified user could infer the Secret GPA values.

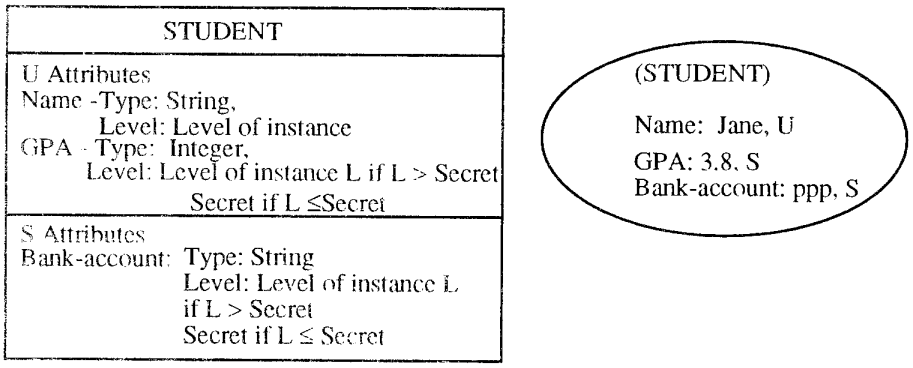


Figure 1 Classification Construct

2.1.3 GENERALIZATION CONSTRUCT

Generalization is the relationship between an object-type and specialized cases of this object-type. These specialized cases are called subtypes. It results in an object type hierarchy also referred to as the IS-A hierarchy. An object-type may have multiple subtypes associated with it. For example, an object-type PERSON could have subtypes STUDENT and EMPLOYEE. PERSON is called the supertype of both STUDENT and EMPLOYEE. The properties of PERSON are inherited by both STUDENT and EMPLOYEE.

---

[2]Note that by the level of an attribute we mean the level of the existence of the attribute. The level of the value is actually the level of the association between the attribute and its value

[3]Note that by the level of the value of an attribute we actually mean the level of the association between the attribute and its value. For example, if the salary value is 20K, then it is meaningless to assign a level to 20K. A security level is assigned to the association between the salary attribute and its value which is 20K.

[4]We use the letters U, C, S, and TS for Unclassified, Confidential Secret, and TopSecret, respectively.

The generalization and classification constructs seem to address the same problem. However, the generalization construct specifies a "top-down" structure, that is given a large set of objects, how can they be decomposed into smaller sets? The specialization construct is basically "bottom-up", that is, given a single object what larger set is it an instance of? This distinction is important for inference analysis and will be discussed later.

Now, in MKDM, the issue is, what should the relationships be between the level of an object-type and the level of its subtype? It is felt that subtyping could be used to protect the more sensitive attributes. Therefore, the property that is enforced is one where the level of a subtype dominates the level of the object-type. This is illustrated in figure 2, where the EMPLOYEE subtype is Secret while the PERSON object-type is Unclassified. If this is not the case, then from the information about the subtype, one could infer the more sensitive information about the object-type.[5]

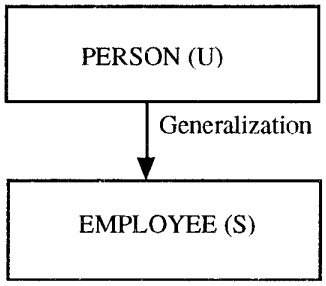


Figure 2. Generalization Construct

The next question is, what about inheritance? It is assumed that everything in the supertype is inherited by the subtype. However, if the level of the attributes of the supertype are lower than the level of the subtype itself, then these inherited attributes of the subtype are assigned the level of the subtype. That is, if EMPLOYEE inherits the name attribute of PERSON, the name attribute of EMPLOYEE is Secret. Multiple inheritance, which occurs when a subtype has multiple supertypes, has interesting consequences with respect to the inference problem. We discuss some issues in section 4.[6]

2.1.4 AGGREGATION CONSTRUCT

The aggregation construct models the IS-PART-OF relationship. For example, a student text book could consist of several chapters. Each chapter may consist of a collection of paragraphs. The question is, what should be the relationship between the level of an object and the level of its components? Can a Secret book have Unclassified chapters or can an Unclassified book have Secret chapters?[7] A flexible model should support both. However, one needs to assign appropriate levels so that security violations via inference do not occur. For example, if the existence of a chapter is to be kept Secret, then one cannot classify the association between the composite attribute and its value for that particular component chapter at the Unclassified level.[8] Otherwise the existence of the chapter would be inferred at the Unclassified level. Figure 3 illustrates a composite object book whose chapters are A, B, and C. The composite attribute as well as the existence of all of its components are Unclassified. The association between the attribute and chapters A and C is Unclassified while the association between the attribute and chapter B is Secret. That is, an Unclassified user knows that there are three chapters, but he can only read chapters A and C.

---

[5]Note that our approach to the treatment of classification and generalization is similar to some of the multilevel object-oriented data models such as the ones proposed in [LUNT89, THUR91a]. Other issues with generalization include single inheritance and multiple inheritance. We adopt the approach proposed in [SELL93].

[6]Also, it should be noted that a subtype could inherit the same attribute at different levels from different supertypes. Conflict resolution rules need to be applied here. We assume that the level of inherited attribute is the least upper bound of the levels involved.

[7]Note that we mean the existence levels of the object and its components.

[8]Note that the value of the composite attribute of a book object would be the chapters of the book. If the association between this attribute and its value is classified at level L, then anyone at level L or higher could read the chapters of the book.

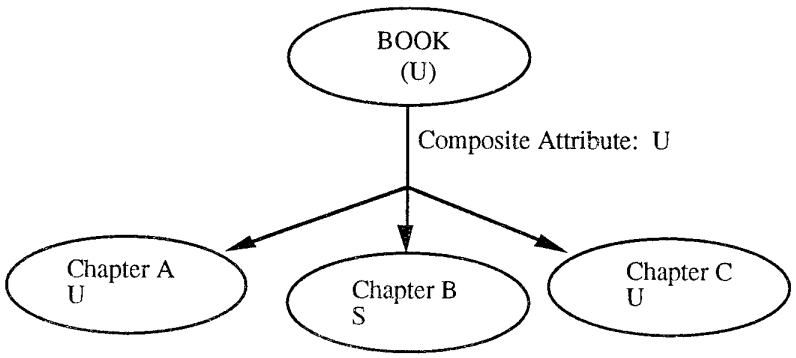


Figure 3. Aggregation Construct

2.1.5 MEMBERSHIP CONSTRUCT

The membership construct is used to model collection objects. A collection object is an object which consists of members which are objects. Each collection object would belong to an object-type. For example, the object-type (TEACHER, STUDENT) is a collection object type whose members are the (teacher, student) pairs. That is, TEACHER and STUDENT and member types of the type (TEACHER, STUDENT). If John is the teacher of Mary, then (John, Mary) would be an instance of the collection object-type (TEACHER, STUDENT).

It must be ensured that the level of a collection object-type must dominate the levels of the object-types which form the collection. Otherwise, one could infer the more sensitive member object-types. The same rule applies to collection objects also. That is, the level of a collection object instance must dominate the levels of the instances which from the collection. That is, if the collection object (John, Mary) is Secret, then the fact that John is a teacher and Mary is a student must be atmost Secret. In some cases, one could classify the fact that (John, x) is Unclassified, but (John, Mary) is Secret. This means that John teaches someone, but the object whom he teaches is not known at the Unclassified level. Suppose one were to have a constraint that the students of John should not be released at the Unclassified level, and if the collection object (John, Mary) is Unclassified, then there is a security violation via inference.

2.1.6 GENERAL CONSTRAINTS

General constraints are constraints enforced on the objects, their properties, and object-types as well as constraints enforced on the relationships between such constructs. Examples include, "the number of courses taken by a senior student should not be less than 2" or "if a student's GPA is less than 2.0, then he cannot take course 333."

Enforcing such constraints across security levels is of concern. For example, the grade of a student for certain subjects could be Secret while for other subjects it could be Unclassified. So, if a constraint such as "if a student's GPA is less than 2.0, then he cannot take course 333" is enforced at the Unclassified level, and if some of the grades are Secret, then if the student's GPA is considered for all of his classes, then there is a possibility for someone at the Unclassified level to infer the grades which are Secret. If only the grades at the Unclassified level are taken into consideration, then there would not be an inference problem, but integrity could be violated. For example, with the GPA of the student when considering only the Unclassified grades could be 3.0 while the actual GPA of the student could be 1.8.

There is a trade-off between preventing the inference problem and maintaining integrity when general constraints are enforced across security levels. In such a situation, the constraints must be considered on a case by case basis. In the case of the GPA example, the consequence of violating integrity may not be catastrophic. So, the inference problem could be given higher priority. In the case of an aircraft example, where the maximum load carried cannot exceed certain weight, and if the cargo are classified at different levels, then violating integrity could be a serious problem if the aircraft is to be able to function. A good discussion of these security problems is found in Marks et al [MARK94].

2.1.7 HEURISTICS

An example of a heuristic constraint is the inference rule that if a student is in his senior year, then he must have taken at least 20 courses. Another example is that if a lecturer teaches more than 2 courses a semester, then he has no external funding to do research. That is heuristics are used to express common knowledge. In formulating heuristic constructs one has to be careful that security violations via inference do not occur. For example, if the fact that a student has taken at least 20 courses is to be kept Secret, then one cannot assert at the Unclassified level that a student is in his senior year. Otherwise, via the heuristic constraint listed above, one could infer at the Unclassified level that the student must have taken at least 20 courses.

Heuristic constraints are in general expressed in the form of IF-THEN statements. They could be complex logical formulas. An example of a more complex constraint is "if a student is in his senior year or a student is in a gifted program, then he must have taken at least 20 courses or his GPA must be at least 3.5. During the inference analysis process, the heuristic constraints as well as the security constraints are examined and if there could be potential inference problems, then the constraints are modified accordingly.

2.1.8 TEMPORAL RELATIONSHIPS

An example of a temporal constraint is that before a student starts his thesis he must finish his qualifying exams and his oral exams. Some of the discussion under heuristics also applies to such constraints. For example, if the fact that the student has either finished his qualifying exams or he has passed his oral exams is to be kept Secret, then the fact that the student has started his thesis cannot be Unclassified. If not, one could infer the Secret information at the Unclassified level.

Inference analysis could also be carried out on the various activities. For example, a student studying for his qualifying exams and a student preparing for his oral exams are both activities. Activities could themselves be assigned security levels. Now, one of these two activities could be carried out at the Unclassified level. But one could enforce the constraint where a student carrying out both activities should be kept Secret until the student starts his thesis. In this case, one of the two activities should be Secret until the student starts his thesis. So, if both activities are assigned the Unclassified level, and the student has not started on his thesis, then during inference analysis such a problem should be detected.

2.1.9 SECURITY CONSTRAINTS

As stated earlier, security constraints are a special type of construct. They are used in the inference analysis process during the modeling of the application. That is, every construct is affected by security constraints. The various types of constraints that may be enforced are:

(i) Constraints that classify an object-type, an object, property, or value
(ii) Constraints that classify the value of an object's property depending on the value of some property.
(iii) Constraints that classify any object-type, object, property, or value depending on the occurrence of some real-world event.
(iv) Constraints that classify associations between collections of attributes and their values.[9]
(v) Constraints which classify activities.

2.2 MULTILEVEL KNOWLEDGE DATA LANGUAGE

Associated with MKDM described in the previous section is an informal specification language that we have developed called Multilevel Knowledge Data Language (MKDL). Our objective was to develop a language which can specify all of the constructs of MKDM as well as translate them into other specification languages such as logic and SQL statements without much difficulty. The language consists of only one construct, the object-type specification. That is, each object-type is specified by an object-type specification where all of the constructs associated with the object-type, such as attributes (properties), subtypes, supertypes, constraints, etc., are specified.

---

[9]For example, the association between (name, value) and (GPA, value) pairs is assigned a level.

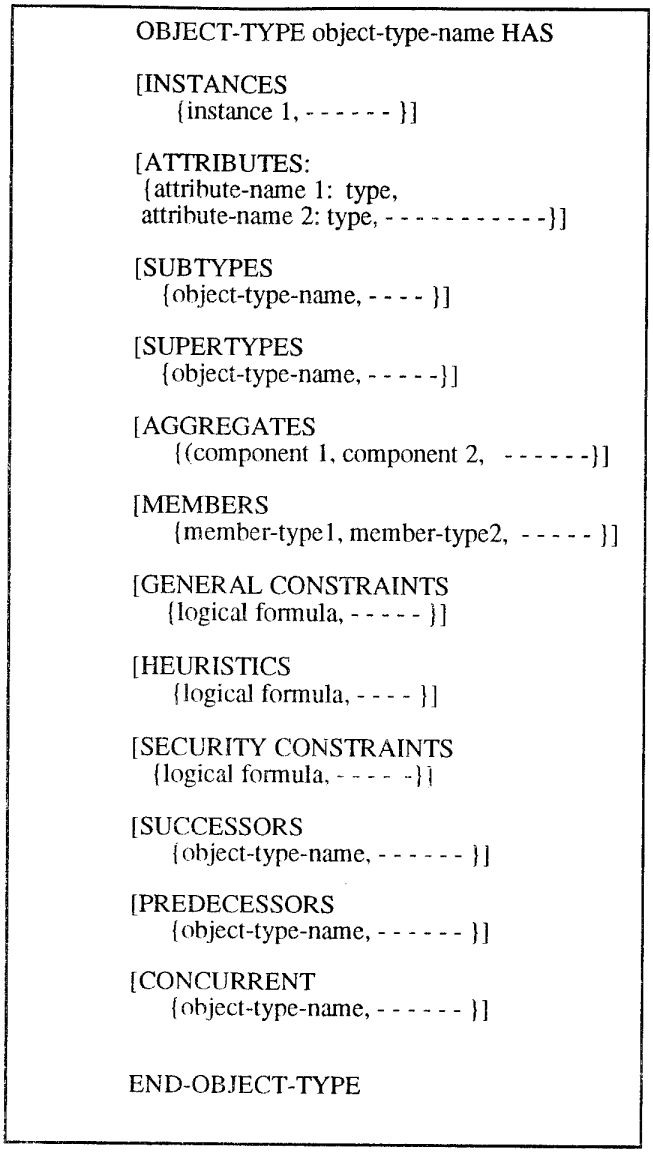


Figure 4. Object-Type Specification

The format of an object-type specification is illustrated in figure 4. The object-type specification has object instance, attributes, subtypes, supertypes, aggregates, members, general constraints, heuristics, temporal relationships specified by successors, predecessors, and concurrents, and security constraints. Not all of the object-type specifications have entries for all components of the specification. For example, the object type STUDENT may not have entries for successors, predecessors, or concurrents. An object type TAKES-COURSE could have entries for successors to indicate the courses to be taken after the completion of the current courses, predecessors to indicate the courses that should have been taken before taking the current courses, and concurrents which specifies the actions, such as working on a project, the student could take in conjunction with the current courses.[10]

[10]Note that aggregate construct specifies the component types of an object if it is a composite object. Member construct specifies the types of the members if the object type has members. Also, constraints are part of the specification of the object type.

Figure 5 illustrates the object type specification of the particular object-type STUDENT. It has three instances s1, s2, and s3. The attributes are names, advisor, enrollment, and GPA. The subtype is GRADUATE-STUDENT while the supertype is PERSON. The general constraint enforced is "maximum number of courses in an enrollment is four."The heuristic constraint is "if the advisor of the student is the Dean, then the student must have a GPA of 3.8 or higher. The security constraint classifies the the association between the GPA attribute and its value at the Secret level. The reason for such a security constraint is to protect the association between the GPA attribute value and the name attribute value. As can be seen STUDENT has no specification for member, temporal, and aggregate constructs.

```
OBJECT-TYPE STUDENT HAS

[INSTANCES
    {s1, s2, s3, - - - - - - }]

[ATTRIBUTES:
 {Name: String,
 Advisor: FACULTY,
  Enrollment: SET-OF-COURSE,
  GPA: Real,
  - - - - - - - - - - -}]

[SUBTYPES
   {GRADUATE-STUDENT, - - - - }]

[SUPERTYPES
   {PERSON, - - - - -}]

[GENERAL CONSTRAINTS
    {Maximum number of courses in
  enrollment is four, - - - - - }]

[HEURISTICS
    {If the advisor of the student is the
Dean, then the student must have a GPA of
3.8. or higher, - - - - }]

[CLASIFICATION CONSTRAINTS
     {student's GPA is Secret,
     - - - - - - - - - }]

END-OBJECT-TYPE STUDENT
```

Figure 5. Object-Type Specification for STUDENT

One of the drawbacks with the specification language that we described earlier in this section is that all of the security constraints are stated under the construct "SECURITY CONSTRAINTS."In fact, some of these constraints could be attached to the other constructs. For example, a particular attribute value could be Secret and this information could be attached to the attribute construct. That is, in addition to the type of the attribute, its security level (i.e. its existence level) and the association between its level and its value are also specified. Such an alternate representation language for OBJECT-TYPE as well as specification for a STUDENT object-type are illustrated in figures 6 and 7, respectively.

As can be seen, associated with object-type, instance, attributes, subtypes, supertypes, etc. there is a security level. Attached to the level, one could also have explanation as to why such a level is assigned. In the specification for STUDENT, there is no entry for SECURITY CONSTRAINTS as levels are attached to the

various constructs. Note that if the constraints are complex logical formulas which consist of many subclauses, they could be specified under SECURITY CONSTRAINTS.

```
OBJECT-TYPE object-type-name HAS
(Level)

[INSTANCES
    {instance 1 (Level), - - - - - - }]

[ATTRIBUTES:
 {(attribute-name 1:  type, (Level), (value Level))
 (attribute-name 2: type, (Level), (value-Level))- - - }]

[SUBTYPES
   {object-type-name (Level), - - - - }]

[SUPERTYPES
   {object-type-name, (Level) - - - - -}]

[AGGREGATES
 {(component 1 (Level), component 2, (Level) - --}]

[MEMBERS
{(member-name (Level), member-type (Level), - - }]

[GENERAL CONSTRAINTS
   {logical formula, (Level)- - - - - }]

[HEURISTICS
    {logical formula, (Level)- - - - }]

[SECURITY CONSTRAINTS
  {logical formula, (Level) - - - - -}]

[SUCCESSORS
    {object-type-name, (Level)- - - - - - }]

[PREDECESSORS
    {object-type-name, (Level)- - - - - - }]

[CONCURRENT
    {object-type-name, (Level)- - - - - - }]

END-OBJECT-TYPE
```

Figure 6. Alternate Object-Type Specification

OBJECT-TYPE STUDENT (U) HAS

[INSTANCES
{s1 (U), s2 (U) , s3 (U), - - - - - - }]

[ATTRIBUTES:
{(Name: String, (U), (U))
(Advisor: FACULTY, (U), (U))
(Enrollment: SET-OF-COURSE, (U), (U))
(GPA: Real, (U) (S) (protect association between GPA and name)) - - - - - - - - - - - -}]

[SUBTYPES
{GRADUATE-STUDENT (U), - - - - }]

[SUPERTYPES
{PERSON (U), - - - - -}]

[GENERAL CONSTRAINTS
{Maximum number of courses in enrollment is four (U), - - - - - }]

[HEURISTICS
{If the advisor of the student is the Dean, then the student must have a GPA of 3.8. or higher (U), - - - - }]

END-OBJECT-TYPE STUDENT

Figure 7. Alternate Object-Type Specification for STUDENT

2.3 GRAPHICAL REPRESENTATION

Since graphical representations are easier to understand by the humans than written specifications, the corresponding graphical representation for the STUDENT object-type specification is illustrated in figure 8. It is basically an extended entity relationship diagram. Each rectangle represents an entity such as a PERSON or STUDENT. Each arrow represents an attribute and points to the entity which is the type of the value of the attribute. The security constraint, the general constraint, and heuristic are specified on the attributes. It should be noted that such a representation may not capture complex general constraints, security constraints, heuristics, members, and temporal relationships. Nevertheless, it can capture the entities, the relationships between them and a reasonable set of constraints. We call the graphical representation based on MKDM to be GRAPHICAL-MKDM

Note that reasoning with the graphical representation scheme is less straightforward than reasoning with a specification language. Therefore, in the modeling process, the first step is usually to represent the entities of the application using a graphical representation scheme. The graphical representation is then translated into some specification language. That is, with MKDM, the first step is to use GRAPHICAL-MKDM and represent the application. Then the representation is transformed in an MKDL specification.

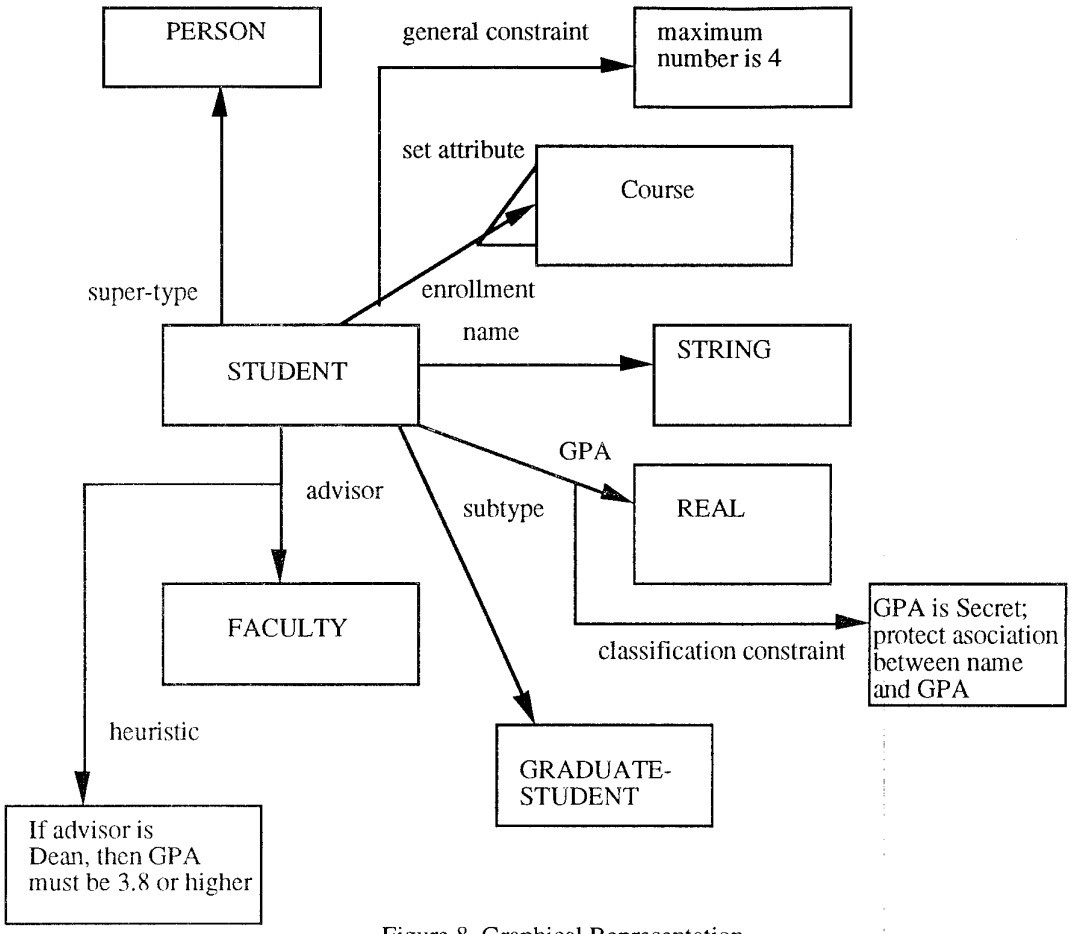


Figure 8. Graphical Representation

3 KNOWLEDGE TRANSFORMATION

3.1 OVERVIEW

As stated in section 1, different representation schemes and inference analysis tools have been developed. Each handles a subset of the inference types and is specialized to particular types of inference problems. That is, the tools are heterogeneous in nature. While standardization has been proposed to handle heterogeneity for operating systems and database systems, it has also been realized that vendors, eager to maintain their advantages over competitors to preserve their share of the market, are not going to abandon their products and develop a single product such as an operating system or a database management system. That is, heterogeneity is here to stay. In the same way, one can expect to see more and more heterogeneous inference analysis tools.

Now, one way to handle heterogeneity is to develop a uniform model of the system which each vendor can interface his product to. That is, in the case of heterogeneous database systems, a global view of the environment can be provided. Consequently, transformations are needed from each local system to the global view of the environment. In the same way, the approach that we have proposed in this paper is intended to give a global view for modeling the multilevel database application and consequently conducting inference analysis. Therefore, transformations are needed from the MKDM methodology proposed here to the various heterogeneous representation schemes such as conceptual structures, logic programming specifications, and SQL specifications proposed by others. This way the transformations can be applied to obtain the individual representation schemes

which means that the inference analysis tools developed for the individual schemes can then be applied. That is, one can take advantage of the complementary tools that have already been developed.

We discuss with a simple example the usefulness of the transformations. Suppose we are given a multilevel database application and the tools developed by Hinke (which is applied to a special conceptual structure-based representation called conceptual graphs) and a tool developed by Binns (which is applied to SQL specifications). Suppose the application designer has difficulty learning about conceptual graphs and SQL, but he is familiar with MKDM methodology. So, he would first represent the application in MKDM related specifications, apply any inference analysis tools developed for MKDM, and then use the transformations to generate conceptual graph-based representation and SQL specifications. Then he would apply Hinke's and Binns' tools for inference analysis. Each tool would uncover a different set of problems and the result would be a more secure design of the application.

This section describes the transformations from the scheme proposed here to conceptual structures, logic programming specifications, and extended SQL specifications. These transformations are described in section 3.2, 3.3, and 3.4.

3.2 CONCEPTUAL STRUCTURES

This section describes how the specifications in GRAPHICAL-MKDM can be transformed into a conceptual structure-based representation. Note that we obtained the graphical representation described in figure 8 from MKDL specifications for the STUDENT object-type. Therefore, one could also transform MKDM specifications into a conceptual structure-based representation.

The particular conceptual structures that we will examine are semantic nets discussed in [THUR90]. We consider a semantic net to be a collection of nodes connected via links. The nodes represent concepts, entities, etc. and the links represent relationships between them. Our treatment of semantic nets is influenced by the work reported in [RICH89]. The entities in GRAPHICAL-MKDM will transform into nodes and the arrows will transform into links between the nodes. Therefore, much of the information in GRAPHICAL-MKDM (except the constraints) will be represented in a similar manner using semantic nets. The constraints such as heuristic and classification constraints, in GRAPHICAL-MKDM will transform into what we have called constraint nets in [THUR90]. For example, the constraint "if the advisor is Dean, then GPA must be 3.8 or higher" is specified using the constraint net illustrated in figure 9.

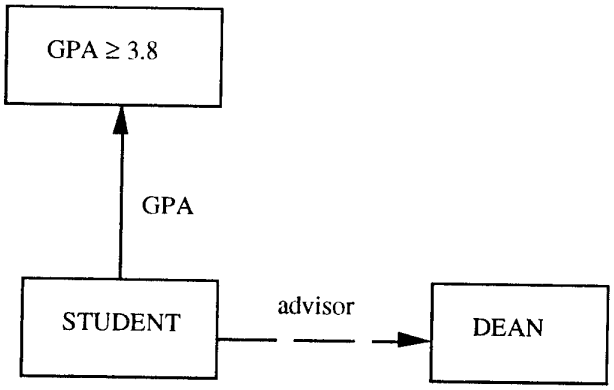


Figure 9. Constraint Net

3.3 LOGIC PROGRAMMING

As stated earlier, while conceptual structures and graphical representations help the humans to capture the essential points of the applications more easily, reasoning with graphical representations could become quite complex. Therefore, specification languages have been developed to specify an application so that analysis tools could be applied on the specification. One of of the popular specification and reasoning languages that has been proposed is based on logic. The advantage of using logic is that it can serve as a specification language or a programming language. Using logic as a programming language enables the programmer to only specify the

application in logic. The programming system will conduct the reasoning and provide the results. This way, the programmer need not be burdened with the details of the procedures.

We have proposed logic programming systems for inference analysis (see for example [THUR89]). The idea is to specify the application as a logic program so that the control component of the program can reason and detect certain inference problems that result due to logical deduction. To use tools based on logic programming, MKDL has to be transformed into a logic-based specification. The specification for STUDENT described in figure 5 will transform into the logic program shown in figure 10. From the first and last clauss shown in this figure, one can deduce that the student s1's GPA is Secret. The program uses backward chaining to make this deduction.

```
STUDENT(s1) <--
STUDENT(s2) <--
 STUDENT(s3) <--
ISA(GRADUATE-STUDENT, STUDENT) <--
ISA(STUDENT, PERSON) <--
 ATTRIBUTE(STUDENT, Name, String) <--
ATTRIBUTE(STUDENT, Advisor, FACULTY) <--
ATTRIBUTE(STUDENT, Enrollment, SET-OF-COURSE) <--
ATTRIBUTE(STUDENT, GPA, Real) <--
N ≤ 4  <-- NUMBER(STUDENT, Enrollment, N)
GPA(S) ≥ 3.8 <-- STUDENT(S) and ADVISOR(S, Dean)
Level(GPA(S)) = Secret <-- STUDENT(S)
```

Figure 10. Logic Programming Specification

3.4 EXTENDED SQL SPECIFICATION

Transforming the object-type specifications into a language such as SQL is highly desirable and in many cases even necessary. This is because many of the MLS/DBMSs that exist today are based on the relational data model with facilities for specifying the schemas in SQL. Furthermore SQL is also an ANSI standard language. Since the object type specification has complex constructs, it is not possible to express all of them in standard SQL. That is, extensions to SQL are necessary to specify the constructs. In this section we discuss the generation of extended SQL statements from the object-type specifications.[11]

Each object-type will translate into a table (or relation) specification in extended SQL. The attributes of an object-type will translate into attributes of a relation. Since subtypes and supertypes are not part of the relational model, the relations which correspond to the subtype and supertypes are inserted into the schema specification for subtypes and supertypes. That is, SQL has to be extended to include subtype and supertype relationships. It should be noted that such extensions have been proposed for SQL to support object-oriented constructs. This version of SQL is called Object SQL.[12] Extensions to SQL are necessary to specify constructs such as general constraints, heuristics, security constraints, and the temporal relationships. One option is to specify these as part of the specification of the table as shown in Figure 11. But this would mean more changes to SQL as integrity constraints are specified separately in SQL and are not part of the table declaration. If we want to be consistent with the SQL standard, then the constraints have to be specified separately. Figure 12 illustrates the extended SQL specification for object-type STUDENT.

---

[11]The discussion provided here on extended SQL is preliminary. One of the objectives is to minimize the changes to the SQL standard

[12]One way to implement the subtype relationship in SQL is to have a table for the supertype and a table for the subtype. The primary key of the table for the supertype will also the the primary of the table of the subtype. Some issues have been discussed in [SELL93].

```
CREATE TABLE table-name

[ATTRIBUTES:
 {attribute-name 1: type,
  attribute-name 2: type,
- - - - - - - - - - -}]

[SUBTYPES
   {table-name1, table-name 2, - - - -}]

[SUPERTYPES
   {table-name1, table-name 2, - - - - }]

[AGGREGATE
   {table-name1, table-name2, - - - - }]

[MEMBERS
   {table-name1, table-name 2, - - -}]

[GENERAL CONSTRAINTS
    {logical formula, - - - - - }]

[HEURISTICS
    {logical formula, - - - - }]

[CLASIFICATION CONSTRAINTS
    {logical formula,  - - - - - - - - - }]

[SUCCESSORS
    {table-name, - - - - - - }]

[PREDECESSORS
    {table-name, - - - - - - }]

[CONCURRENT
    {table-name, - - - - - - }]

[TUPLES
    {tuple 1, - - - - - - }]

END CREATE TABLE
```

Figure 11. Extended SQL Specification

As can be seen, the object-types correspond to table names. The instances and attributes of object-types translate into tuples and attributes of tables. SUBTYPES and SUPERTYPES are extensions to SQL that have to be made to support the associated constructs in MKDL. Because these constructs are object-types, they can be specified by table names. The member construct is specified by a collection of table names where each table name in the collection corresponds to a member type. A tuple of the member table would consist of elements where each element is the primary key of a table which corresponds to a member type. Similarly, aggregation construct can also be specified by a list of table names where each table in the list corresponds to the type of a component object.[13] General constraints, heuristics, and security constraints are expressed as formulas and, as stated earlier, they could be specified outside of table declaration or within a table declaration as shown in figure 12. The temporal construct is represented by table-names. Here we assume that information about an activity can be

[13]More research needs to be done on specifying and implementing aggregate objects in the relational model.

represented by a table. Whether such a scheme is sufficient to represent all of the temporal constructs is yet to be determined.[14]

```
CREATE TABLE STUDENT

[ATTRIBUTES:
 {Name: String,
 Advisor: FACULTY,
  Enrollment: SET-OF-COURSE,
  GPA: Real,
   - - - - - - - - - - - }]

[SUBTYPES
   {GRADUATE-STUDENT, - - - - }]

[SUPERTYPES
   {PERSON, - - - - - }]

[GENERAL CONSTRAINTS
    {Maximum number of courses in
enrollment is four, - - - - - }]

[HEURISTICS
    {If the advisor of the student is the
Dean, then the student must have a GPA of 3.8, or
higher, - - - }]

[CLASIFICATION CONSTRAINTS
{(student's GPA is Secret, protect the association
between the attributes GPA and name )
      - - - - - - - - - }]

[TUPLES
    {s1, s2, s3, - - - - - - }]

END CREATE TABLE STUDENT
```

Figure 12. Extended SQL Specification for STUDENT

4. INFERENCE TYPES AND INFERENCE ANALYSIS

GRAPHICAL-MKDM and MKDL capture sufficient semantics of the application so that inference tools can be developed for them. Furthermore, they are sufficiently general enough so that they can be transformed into existing representation schemes such as conceptual structures, logic programming, and extended SQL. This way, existing inference tools could be applied to the different specifications. Note that some aspects of inference analysis was given in section 2 when we described hypersemantic data modeling. This section describes inference analysis in more detail. Before one applies inference analysis tools, one needs to determine what types of inference are to be handled. In section 4.1 we will describe some of the inference types that we are interested in. In section 4.2 we discuss inference analysis.

---

[14]Note that instead of specifying all of the security constraints under the SECURITY CONSTRAINTS construct, as illustrated in figures 11 and 12, one could attach levels with the tables and the attributes whenever possible. For example, if the attribute GPA is Secret, then next to this attribute, a label S could be attached. That is, extended SQL specifications which correspond to figures 6 and 7 can also be given.

4.1 INFERENCE TYPES

In section 4.1.1 we discuss some inference types that are more special to (but not necessarily limited to) MKDM and in section 4.1.2 we discuss some of the more general inference types.

4.1.1 INFERENCE TYPES FOR MKDM[15]

The Multilevel Knowledge Data Model provides a natural means to control some inference in databases. Inference problem occurs when a Low cleared user, retrieving Low classified data, is able to *infer* High classified data. Such inference capabilities are not part of the database mechanism, but instead depend upon the semantic and logical relationship of the data. For the inference problem to occur, data is accumulated into a meaningful *concept*, and knowledge about the instance of that concept is then applied to derive additional attributes. For example, the object (name, salary) might be classified High, but the two objects (name, job) and (job, salary) might be sufficient to infer the restricted object (name, salary). This will be true if there are two knowledge rules, "each person has exactly one job" and "each job has exactly one salary" which hold for all individual instances of the concept.

The simple classification constraints might be sufficient to address the inference problem if all instances in the database belonged to a single object-type, but problems arise when an object inherits from multiple object-types. In many cases, real objects belong to multiple object types. For example, the object- types Hospital_patient and Hospital_employee may have some identical instances, those employees who are also patients. The object-types, however, could be maintained by independent departments (admitting and personnel) with patient data being Confidential and employee data being Unclassified. Those instances belonging to both would have to be classified at the highest of these levels, i.e. Confidential. Raising the classification of those instances means that the Unclassified users of Hospital_employee can no longer access that information. One solution to this problem is to keep the object-types separate, without shared instances. In this case, those employee-patients will have two objects with some information (such as name, social security number) being duplicated.

However, as the (name, salary) example illustrates, duplicating information may increase inference threats. Inference is frequently accomplished by retrieving objects from different object-types and using smaller "pieces" of these objects to form the restricted object. That is, *name* is retrieved from one object, and *salary* from another. At no time does the user retrieve an instance from the classified object-type, yet such an instance results from combining unclassified objects. We need a way to verify that a classified object-type has been created. To solve this, we adapt the following definition:

An object *is an instance of* an object-type if the object has those attributes specified by the object type.

(This definition has been paraphrased in the real world as "if it walks like a duck, and it quacks like a duck, then it is a duck"!)

We can now define an object-type in terms of its instances. We will call such things virtual objects. A virtual object is any collection of instances having a common set of attributes defined in database. Virtual objects will be the means by which we can keep track of user's access to parts of restricted objects. We now discuss some of the inference types for MKDM.

CASE 1. Sub-types Accumulate to Release Object-type

If we have an aggregation construct (IS-PART-OF), inheritance issues do not apply. Release of the object does not necessarily release the "parts", however we must assume that release of a "part" will release some information about the object. Such information may or may not be sufficient to compromise the object. *Release of all parts is assumed to release all of the object.* If the object has a higher classification than the part objects, then the object must be assigned a *threshold*, and only a limited number of parts may be released to a lower cleared process. Note that a similar argument can be applied for the relationship between an object-type and its part (or component) object-types.

---

[15]Our discussion of inference types for MKDM is preliminary. We feel that an investigation of the inference problem for the object-oriented model warrants a detailed investigation.

The next two inference types are related and deal with relationships among real and virtual object-types. Rather than try to account for all possible virtual object-types, we will show those virtual object-types derivable from individual object-types and then show how these may be composed into additional virtual object-types. The structure of an object-type immediately leads to a virtual object structure for sub-types. For example, let A be a generalization construct consisting of sub-types A_1, A_2, ...A_n. Each A_i isa A. Since A_i isa A, each instance of A_i inherits all the attributes of A, plus possibly some additional ones. Instances in A_i *restricted to* the attributes inherited from A therefore define a virtual object-type, denoted A_i | A. The instances of A_i | A have identical attributes, (those specified by A) and belong to the generalization class denoted by A. We can say that A_i | A isa A. If A_i itself has subtypes, these will inherit all of the A_i attributes as well as all of A's attributes, and hence may be used to define two virtual object-types. These particular virtual object-types follow the transitivity relation as do regular sub-types.

Transitivity.
If A isa B and B isa C, then A isa C.
If A | B isa B and B | C isa C, then A | C isa C.

CASE 2. Object-type Releases Virtual Sub-types

Let us consider the generalization construct for object-type PERSON, having sub-types STUDENT and EMPLOYEE. If all instances of PERSON are known, at least some attributes (those relating to PERSON) of all instances of both STUDENT and EMPLOYEE are also known. Release of all instances of PERSON therefore releases all instances in the virtual sub-types STUDENT | PERSON and EMPLOYEE | PERSON. Since sub-types must be classified at least as high as the parent object-type, this does not directly compromise any information. However, information on such disclosures must be maintained since they may be combined with other methods, such as case 3 below.[16] This observation may be summarized as:

Lemma 1: If we retrieve all the instances of object-type A, and B isa A, we will be able to infer all the instances of the virtual object-type B | A.

CASE 3. Instance in Virtual Sub-type Releases Instance in Object-type

Case 2 applies to cases where we release all instances of some object-type. Suppose, however, that we release only one instance, does this lead to an inference problem? Now, each instance of a sub-type is an instance of the parent object-type. Instances in a sub-type inherit all the attributes of the parent, so this is restated as:

Lemma 2: If we retrieve an instance in object-type A, and A isa B, we will be able to infer an instance in object-type B.

These two lemmas are utilized in the following theorem:

Inference Theorem: If we retrieve instances of object-type A, we will be able to infer instances of object-type B if B | A isa B, and B | A $\neq \emptyset$.

Proof: By lemma 1 retrieving an instance of object-type A allows us to infer an instance of object-type B | A. If B | A isa B, then by lemma 2, we can infer an instance of B for each instance of B | A. The instances found in B may not be unique, however, and we cannot guarantee that all instances of B are derivable from instances of A.

Informal Description of the Inference Theorem: If we retrieve instances of object-type A, we will be able to infer instances of object-type B if the attributes of B are a subset of the attributes of A.

Example: Assume that a company has a classified contract, i.e. the people working on contract X may not be retrieved by uncleared database users. Now assume that the database for this company consists of object-type COMPANY_EMP, containing the names and SS# of all the employees, and object-types (sub-classes) ENGINEER, and SUPPORT containing the appropriate attributes. Now assume a second, classified object-type

---

[16] For example, there could be a case where one infers the instances of a classified object-type C from the instances of the Unclassified object-types CIA and CIB.

PROJECT_X_EMP, containing only the names of the employees working on project X. Suppose we retrieve instances of object-type ENGINEER. In the inference theorem, ENGINEER then corresponds to object-type A and PROJECT_X_EMP corresponds to object-type B. The only attribute common to both objects is name, (attributes of B | A = {name}) which is identical to the attribute set of object-type B and hence B | A isa B. So release of unclassified instances of object-type ENGINEER releases names of employees in classified object-type PROJECT_X_EMP.

The difficulty in inference control comes in applying the inference theorem. Database models typically do not specify all possible generalization relationships. It may be possible to access several objects and form a chain of inferences to compromise a classified object. In the worst case, all possible combinations of all objects must be considered as possible inference paths.

Much of the current research in inference control centers around finding and specifying the relationships between objects, especially objects that actually belong to object-types but are not specified in the object-type definition. Such hidden relationships may be combined to form an inference *path*. The work of Binns [BINN92] and Garvey [GARV92] address the relational equivalent of finding these types of chains, or paths. The conceptual structures used by Thuraisingham [THUR90] provide a graphical method of defining these relationships between objects. Hinke [HINK92] has developed a knowledge engineering tool designed to assist the database designer in defining relationships between such data concepts. Ideally, these methods would enable us to specify all the generalization relations between all possible objects, and the inference relationships would then be evident.

The use of the object-oriented paradigm offers certain benefits over the relational model. In particular, arbitrary combinations of attributes need not be considered. Each object has a predefined set of attributes, and new combinations of attributes which are not derivable from the existing predefined sets need not be examined. The question now becomes: what new, virtual objects need to be controlled? Of course, we must prevent the construction of any high classified object from lower classified pieces. Some such objects will be explicitly defined via the classification constraints. That is, the classification constraints may be viewed as specifying a classified virtual object or object-type.

An additional consideration for controlling deductive inference comes from considering the constraints. General and heuristic constraints are a means of specifying connections between objects that are not derivable from the specified hierarchical structure. They represent an additional set of deductive rules that may need to be handled by use of logic programming and is discussed in section 4.2.2. It is important to realize, however, that the techniques are not separate but complement each other.

4.1.2 OTHER INFERENCE TYPES

Some of the more common inference types that have been discussed in the literature include inference by logical deduction and semantic association. For example if A implies B, A is Unclassified, and B is Secret, then there is an inference problem through logical deduction. As another example if A and B are unclassified individually, but taken together they are Secret, then there is an inference problem through semantic association. Note that the inference problem through semantic association is in many ways similar to the inference problem that results from the aggregate hierarchy. A discussion of some of the other inference types is given in [THUR91b].

4.2 INFERENCE ANALYSIS

As described earlier, inference analysis tools can be applied on GRAPHICAL-MKDM, MKDL, or on transformed specifications such as conceptual structures, logic programming specifications, and extended SQL. We describe some of the essential points in this section. Note that inference analysis has to be a repetitive process if the application under consideration is a dynamic one. For example, new entities and relationships could be introduced and the security levels of the entities could also change. Therefore, at every step, the various analysis tools have to be applied to prevent security violations via inference.

4.2.1 TOOLS APPLIED TO THE HYPERSEMANTIC MODEL-BASED REPRESENTATION

This section describes some inference analysis tools that could be applied on the hypersemantic model-based representation that we have discussed in section 2.

Consider the example of an inference analysis tool which could be used on the specification in MKDL to detect certain inference problems. The security constraints could be applied to each modeling construct such as classification, generalization, and temporal relationships. If it is detected that there is a security violation via inference as discussed in section 2, then the designer is notified and the security levels assigned to the various constructs are adjusted. For example, if the subtype is assigned a lower level than the object-type, then the security rule for the inheritance hierarchy is violated, and the designer is notified of the problem. As another example, heuristic rules can be used to deduce derived information and the security constraints could be used to assign the security levels to the derived information. If the level of the derived information is higher than the level of the information used to derive this information, then there is a potential for security violation via inference. Such logical inferences could be detected during the inference analysis process.

Tools to handle some of the inference types discussed in section 4.1.1 are quite complex and require further research. We are conducting some preliminary research toward designing a tool based on a graphical model which shows how one object may be used to derive another object. Some of the issues toward developing such a tool were discussed in section 4.1.1. Using such a tool, one could deduce whether a higher level object could be derived from lower level objects.

4.2.2 APPLYING INFERENCE ANALYSIS TOOLS ON TRANSFORMED REPRESENTATIONS

This section describes inference analysis on the transformed representations. Since the transformed representations are essentially those developed by others, the information in this section is taken from the inference work published by others. Also, since we have focussed mainly on representations based on conceptual structures, logic programming, and SQL, we will describe the analysis tools designed for such representations.

Reasoning with conceptual structures for inference prevention has been described in [THUR90]. Some tools based on a similar approach have been developed by Garvey et al [GARV92] and Hinke et al [HINK92]. With conceptual structures one could use a variety of inference rules for deductions. These include transitive rule, distribution rule, and pattern matching. For example, with if there is a net which asserts that champion isa ship and ship has a captain, then one can deduce through transitive rule that champion has a captain. If one wants to protect that fact that champion has a captain at High and the information in the net is Low, then there an inference problem. Pattern matching is one of the inference rules used in semantic nets to derive new information from the main net and the constraint nets. For example, consider the constraint net of figure 9. If in the main net there is an arrow from STUDENT to DEAN, then one can conclude that the student's GPA must be 3.8 or higher.

Logic programming techniques are used for inference detection in the following manner. As discussed in section 3, one specifies the application as a logic program. As new clauses are added to the program, they are tested as queries to see if there is an inconsistency. As a simple example, suppose the program consists of the following clauses.

Level(X, Secret) <-- TEACHER(John, X)
Level(Mary, Unclassified) <--
NOT Level(X, Secret) <-- Level(X, Unclassified)

They assert that all those who learn from John are Secret and that Mary is Unclassified. Furthermore, the third clause asserts that any entity which is Unclassified cannot be Secret. Suppose one wants to assert that John teaches Mary. This is now tested as a query. That is, the following query
<-- TEACHER(John, Mary)
is posed. Through backward chaining, a contradiction is derived. This means that if John were to teach Mary, there will be an inference problem.

The extended SQL specification generated could be used to apply the inference analysis tool developed by Binns [BINN92]. Binns' tool uses a technique called secondary path analysis. It is assumed that an attribute of a relation is classified in order to protect a collection of attributes which includes the classified attribute. For example, if the grade attribute of a student is classified at the Secret level, it is assumed that it is classified in order to protect the grade associated with some other attribute of the same relation or possibly a different relation. A graph structure is used to represent the paths between the various attributes of relations. Paths are obtained by

performing the join operation between the relations. The lines forming the path are classified according to the classification constraints specified in the schema. If there is a path between two attributes where some part of it is Secret, and if there is a completely Unclassified path between the same two attributes, then there is a potential for an inference problem. The tool could point out such problems. To apply Binns' tool, the extended SQL specifications discussed in section 3 are necessary.

5. RELATED WORK

As stated in section 1, several proposals on using conceptual structures for representing and reasoning about multilevel database applications have been given. We provide a brief overview of the various efforts and compare the approach proposed in this paper with the others.

To our knowledge, the use of conceptual structures to handle the inference problem was first proposed by Burns [BURN88] and Hinke [HINK88]. While Burns proposed the use of the entity-relationship model, Hinke's work was on the use of graphs for representing the application. He showed how inferences may be detected by traversing alternate paths between two nodes in the graph. Further work on the use of conceptual structures for inference handling was proposed by Smith [SMIT90]. Smith suggests extensions to the semantic data model discussed in [URBA89] to represent multilevel applications. While he has shown how the the model could be used for representation, reasoning techniques are not addressed. Thuraisingham [THUR90] showed how conceptual structures such as semantic nets and conceptual graphs could be used to represent and reason about the multilevel database application.

More recently the development of tools have been reported by Binns, Hinke, and Garvey et al. Binns has developed a tool for secondary path analysis. As stated in section 5, this tool takes as input SQL specifications and generates modified specifications. Hinke has developed a tool called AERIE which is based on conceptual graphs. Garvey et al. have developed a tool based on semantic nets. At present, Collins [COLL94] is developing an inference analysis tool using CLIPS.

While the approaches described above have focused mainly on the inference problem, a more general approach for multilevel database application design has been reported in [WISE91, PERN92, and SELL93]. The approach, particularly in [PERN92] and [SELL93], is not only to capture the structural aspects of the application, but also the dynamic aspects of the application. The goal is to design the multilevel database and the automated system. While the inference problem has been given some consideration, it is not the major focus.

As stated earlier, the approach proposed in this paper focusses on developing a uniform representation scheme that can be transformed into other representation schemes without much difficulty so that the inference analysis tools developed could be applied so that one can obtain the maximum benefit from the tools that are already available. One could also develop inference analysis tools for MKDM. The major contribution of MKDM is that it incorporates constructs from data models as well as knowledge models. Therefore, it encompasses the essential capabilities of the previous models discussed in the literature such as the ones developed by Burns, Garvey, Hinke, Smith, Thuraisingham, and others. The paper also gives a specification language and a graphical representation of the model. Since MKDM borrows constructs from the data and knowledge models, the translation of MKDM and MKDL into other representation schemes such as conceptual structures, logic programming specification, and extended SQL can be accomplished without much difficulty. In summary, the strength of the approach proposed in this paper is its generality.[17]

[17]One could argue that generality has some disadvantages in that one may not be able to get the full potential of a single tool. The question of generality vs speciality has been discussed in other fields such as heterogeneous data/knowledge base systems integration. The ultimate decision would depend on what the client wants. That is, should one be able to use a collection of inference analysis tools in a reasonable manner or get the maximum benefit of one tool? Our hope is that this paper will make the community start thinking about addressing some of these issues as inference analysis tools continue to develop.

6. SUMMARY AND FUTURE CONSIDERATIONS

This paper has described a model called Multilevel Knowledge Data Model (MKDM) and an associated specification language called Multilevel Knowledge Data Language (MKDL). MKDM combines constructs both from data models and knowledge models. Because of this, it has the representational power of semantic data models, and the reasoning power of knowledge models. In describing the various constructs of MKDM we also showed how potential security violations could be detected during the modeling process. Next we described how GRAPHICAL-MKDM and MKDL could be transformed into other representation schemes such as conceptual structures, logic programming specifications, and extended SQL. Finally we discussed different inference types and how inference analysis tools could be applied on the various representations. Comparison of the approach described in this paper with other approaches in the literature was also given.

This paper provides the direction for modeling the various entities of multilevel database application, capturing the security semantics of the application, and subsequently applying reasoning tools for inference analysis. Future research should include the following:

(i) Develop inference analysis tools for MKDM. Since MKDM is based on an object-oriented model, we discussed various inference types that can be uncovered with such a representation. Inference analysis tools to detect potential problems with such representations are yet to be designed. In section 4 we discused various aspects of such tools with examples. Tools based on generalized algorithms need to be developed.

(iii) Develop transformations to other representations so that current inference analysis tools can be applied. Some of the techniques for transforming MKDM constructs into other representations such as conceptual structures, logic programming specifications, and SQL were discussed in section 3. Tools for such transformations have to be developed.

(iii) Test the tools with examples. One needs to take a real world application, represent it using GRAPHICAL-MKDM and MKDL, apply the inference analysis tools developed for MKDM, use transformation tools to generate other specifications, and subsequently apply inference analysis tools such as the ones proposed in [BINN92, HINK92, GARV92, COLL94]. Such an exercise would demonstrate the usefulness as well as the robustness of the methodology that we have developed.

ACKNOWLEDGMENT

We thank Dale Johnson for his support. We also thank Rae Burns and Kenneth Smith for comments on this paper.

REFERENCES

[ANSI92] American National Standards Institute, SQL-2 Specification, 1992.

[BINN92] Binns, L., August 1992, "Inference Through Secondary Path Analysis," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[BURN88] Burns, R., May 1988, "ER Approach to Multilevel Database Design," Presented at the 1st RADC Database Security Workshop Menlo Park, CA (Proceedings published by Springer Verlag, 1992)

[BURN92] Burns, R., October 1992, A Conceptual Model for Multilevel Database Design, Presented at the 5th Rome Laboratory Database Security Workshop, Fredonia, NY.

[COLL94] Collins, M., 1994 Design and Implementation of an Inference Analysis Tool using Conceptual Structures, In preparation.

[MARK94] Marks D. et al, "Security Considerations of Content and Context Based Access Controls," Proceedings of SEC '94, IFIP WG11, May 1994.

[MORG87] Morgenstern, M., May 1987, "Security and Inference in Multilevel Database and Knowledge Base Systems," Proceedings of the ACM SIGMOD Conference, San Francisco, CA.

[GARV92] Garvey, T., et al., August 1992, Toward a Tool to Detect and Eliminate Inference Problems," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, BC.

[HINK92] Hinke T., and H. Delugach, August 1992, "Aerie: An Inference Modeling and Detection Approach for Databases," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[LUNT89] Lunt, T., September 1989, "Multilevel Security for Object-oriented Database Systems," Proceedings of the 3rd IFIP Database Security Conference, Monterey, CA.

[PERN92] Pernul, G., December 1992, "Security Constraint Processing During Multilevel Secure Database Design," Proceedings of the 8th Computer Security Applications Conference, San Antonio, TX.

[POTT89] Potter, W. et al. 1989, Hypersemantic Data Modeling, Data and Knowledge Engineering Journal, Vol. 4.

[RICH89] [RICH89] Richards, T., 1989, *Clausal Form Logic: An Introduction to the Logic of Computer Reasoning*. Sydney, Australia: Addison Wesley.

[SELL93] Sell, P. and B. Thuraisingham, September 1993, "Applying OMT for Multilevel Database Application Design," Proceedings of the 7th IFIP Database Security Conference, Vancouver, BC.

[SMIT90] Smith, G., May 1990, "Modelling Security-Relevant Data Semantics," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[THUR89] Thuraisingham, B., 1989, Secure Query Processing in Intelligent database Management Systems, Proceedings of the 5th Computer Security Applications Conference, Tucson, AZ

[THUR90] Thuraisingham, B. M., August 1990, *The Use of Conceptual Structures in Handling the Inference Problem*, Technical Report M90-55, The MITRE Corporation, Bedford, MA (also published in the Proceedings of the 5th IFIP Database Security Conference, 1991).

[THUR91a] Thuraisingham, B. M., 1991, Multilevel Object-Oriented Data Model: Issues on Noncomposite Objects, Composite Objects, Versioning, and Concurrency Control, Journal of Object-Oriented Programming, Vol. 4.

[THUR91b] Thuraisingham, B., 1991, Inference Problem in Database Security, CIPHER Newsletter, Winter 1991.

[URBA90] Urban, S., and L. Delacombre, December 1990, "Constraint Analysis," IEEE Transactions on Knowledge and Data Engineering, Vol. 2.

[WISE91] Wiseman, S., November 1991, "Abstract and Concrete Models for Secure Database Applications", Proceedings of the 5th IFIP Working Conference in Database Security, Shepherdstown, W. VA.

Database models:

Chair: E. Fernandez

Florida Atlantic Uni., FL

A Multilevel Secure Federated Database

Martin S Olivier*
Rand Afrikaans University

Abstract

This paper proposes a model for a multilevel secure federated database. A federated database is a distributed database that is characterised by a high degree of site autonomy, yet the sites cooperate on global transactions.

The proposed model has three main features: (1) it is intended for a loosely coupled federation with almost no central authority; (2) local classification of a data item is honoured by all members of the federation; and (3) a site can decide on the level of sensitivity of its data that may be sent to each other site.

The model solves the problem where the sites are homogeneous; however more work needs to be done for heterogeneous sites.

Keywords: Security, object-orientation, distributed databases

1 Introduction

Distributed databases have some well known advantages; amongst others, access speed and availability of information can be increased since information can be stored in close physical proximity to where it is most often used. On the negative side, distributed databases are more complex than their centralised counterparts.

Few papers have been published that address the security issues specifically relevant to distributed databases. In a series of three papers Thuraisingham *et al* [18, 28, 29] have developed models for multilevel secure relational databases that assume three increasingly complex database architectures. In the first paper [18] the data distribution reflects the classification of the data; however, this database is not a true distributed database [29, p662]. The second paper [28] is based on a true distributed database and assumes that the various databases forming the distributed database are homogeneous. The third paper [29] also uses a true distributed database, but allows for limited heterogeneity: only the classification ranges for data at various nodes are restricted.

Bull, Gong and Sollins[3] argue that security in a federated system should be governed from the servers and not by using conventional access control lists or capabilities. They do not address multilevel security. Gong and Qian[10] show that interoperation of systems can cause unintended (indirect) access to information and prove that elimination of such unintended access for a simplified case is NP complete. One solution is to

*Department of Computer Science, Rand Afrikaans University, PO Box 524, Auckland Park, Johannesburg, South Africa 2006; molivier@rkw.rau.ac.za

achieve secure global interoperation "incrementally by composing secure local interoperation."

Obviously, since a distributed database is a collection of other databases, most of the results obtained for secure centralised databases also apply in the case of distributed databases—see [7, 21] for overviews of centralised security and [11, 12, 13, 15, 20, 27] for examples of models for (centralised) secure object-oriented databases and [16, 17] for a relational example.

We are of the opinion that the additional security requirements posed by distributed databases do depend on the architecture of the distributed database. To illustrate this, consider the classification for such databases described by [22]: they consider (1) the autonomy of the paticipating databases, (2) their (geographical) distribution, and (3) whether they are homogeneous or heterogeneous. As an example, consider a distributed database where the participating sites are homogeneous with little local autonomy. Such a database probably needs very little more for security than a centralised database needs (apart from a secure way to communicate between the participating sites). This applies whether the database is geographically distributed or not. On the other hand, any distributed database that combines heterogeneous participating databases will require a significant amount of additional security facilities to co-operate securely.

It would therefore seem that various categories of distributed databases warrant investigation. For example, the papers by Thuraisingham and others mentioned earlier [18, 28, 29] use different database architectures. Similarly, while the paper by Varadharajan and Black [31] on 'Distributed Object-oriented Databases' does not specifically address any 'distributed' aspects, it does apply to what Özsu and Valduriez [22] call an *integrated database*—a distributed database that consists of homogeneous sites with little local autonomy. Other papers dealing with secure centralised databases (such as those cited earlier) similarly apply to such integrated databases—as long as the described models do not incur exessively high communication costs.

The categories of secure distributed databases that have not received enough attention in the literature seem to be those distributed databases that provide a high degree of local autonomy to the participating sites (the concern of this paper) and the various possibilities that exist for heterogeneous sites. Note that the latter category includes a number of cases, from those where only the security systems differ, through the case where the data models are similar (but not identical) to the distributed databases where the data models need not be the same at the participating sites. As an example, the sites of the model described in [29] only differ in the sensitivity ranges of data that are allowed on each site; one site may for example contain data in the range *restricted* to *top secret*, while another may contain information in the range *unclassified* to *secret*.

This paper proposes a model for a distributed database that allows a high degree of local autonomy. To use the classification of Özsu and Valduriez [22, p81], we are interested in a *federated database*, that is one that does allow a high degree of local autonomy, but where the sites can cooperate on global queries and transactions (compared to a so called *multidatabase*, where the participating sites are autonomous but very loosely integrated). We assume that no central authority exists that can decide on (for example) classification of data and clearance of users. This model will be referred to as SeFD (*Secure Federated Database*).

The required site autonomy of a federated database makes security a fundamental issue of such a database since "site autonomy is achieved when each site is able both to control accesses from other sites to its own data and to manipulate its data without being conditioned by any other site" [4, p323].

SeFD uses multilevel security—that is security where the decision whether a subject should be allowed to access an entity is based on the clearance of the subject and the sensitivity of the entity (and the type of access requested). For example, a subject is often allowed to read the entity if the clearance of the subject dominates the sensitivity of the entity. See section 2.2 for more details.

SeFD assumes that the participating sites are homogeneous and (more specifically) object-oriented. By homogeneity we mean that the participating sites use the same data and security models; to simplify matters one can assume that the DBMSs at the various sites are copies of the same product. In particular, clearance and sensitivity levels at one site will directly correspond to levels at other sites. We make the assumption of homogeneity in order to concentrate on the questions regarding local autonomy—see section 5 for a short reflection on databases where homogeneity is not assumed. The assumption that the model should be object-oriented is not necessary, but has been made because this work forms part of a more comprehensive project dealing with secure object-oriented databases [19, 20, 21]. Many of the comments made, do apply to other database models; however this will not be discussed in the current paper.

The next section contains background material on object-oriented databases, multilevel security and federated databases. Section 3 then considers the security requirements posed by a federated database. Section 4 describes the proposed model. This is followed by the conclusion including a short description of future research.

2 Background

This section briefly introduces the concepts that form the basis of issues used in this paper. These introductions are intended to enable readers not familiar with the concepts to follow the rest of the paper. It is also intended to indicate the particular meanings associated with terms in this paper—especially terms that do not have a univocal meaning in the literature. This section is not intended as a comprehensive treatment of the concepts, or used in defence of this paper's view of the concepts; in particular, where alternatives exist, these alternatives are not pointed out. References are given for those readers who require a more comprehensive treatment.

2.1 Object-oriented databases

Object-oriented databases are databases that use object-oriented concepts to implement the database. The basic unit used to store data is the object. An *object* represents a logically single entity; it is an encapsulated unit consisting of both the data (*instance variables*) and procedural code (*methods*) to manipulate the data. Objects may only be accessed by activating its methods. A method is activated by sending a *message* to it. A method itself consists of a sequence of messages to be sent to objects

(interleaved with operations to read and write instance variables). Often reading and writing of instance variables are modelled as messages sent to (and replies received from) instance variable 'objects.' A database request is initiated by a user (or application program) sending the first message; the remainder of the request then consists of a sequence of messages sent between objects.

Objects are instantiated from classes—we assume that a *class* is an object itself that serves as a template for the *instances* (objects) of that class. Classes can be derived from other classes by adding variables and/or methods. A class thus derived is known as a *subclass* of the other class, while the original class is known as the *superclass* of the derived class. The process where the subclass uses the same declarations for variables and/or methods as its superclass is known as *inheritance*.

We assume that all data items in the model are objects—this includes classes and 'primitive' items such as integer variables. This assumption is also made by Smalltalk [9] that serves as our model of object-orientation.

See [1, 2, 14] for a description of object-oriented databases, [5] for a comprehensive treatment of databases in general and [32, 33] for a description of the object-oriented paradigm.

2.2 Multilevel security

In a secure system, requests to access resources are allowed or disallowed depending on security criteria. The possible issuers (originators) of requests are usually referred to as *subjects*. The resources accessed by the request are usually referred to as *objects*; however, in this paper the term *entity* will be used to refer to the target of a request and the term *object* will be used exclusively in its object-oriented sense.

The criteria used to decide whether a subject should be allowed to access an entity are usually divided into two categories: In discretionary security, entities are owned by specific subjects; such a subject then has the discretionary power to grant other subjects access rights to its entities (and to revoke such rights from other subjects). Multilevel security (or mandatory security) refers to a system where all subjects are grouped into categories; similarly all entities are grouped into categories, and then it is indicated which categories of users are allowed to access entities in any given category. This is often accomplished by assigning *clearance* labels to subjects and *sensitivity* labels to entities and then only allowing a subject to access an entity if a specific relationship holds between the clearance of the subject and the sensitivity level of the entity. For example, a subject is often allowed to read the entity if the clearance of the subject dominates the sensitivity of the entity. In contrast, a subject is usually allowed to write to the entity if the clearance of the subject is dominated by the sensitivity of the entity. Further, access restrictions remain in place, even if the information is copied or otherwise manipulated. In contrast to discretionary security where individual subjects have the authority to grant access to other subjects, security classifications in a mandatory secure system are determined by a particular individual (or group) with this responsibility for the entire database. This individual (or group) is often known as the system security officer.

See [23, pp285–286] for a discussion of discretionary security and [23, pp329–340] for a discussion of multilevel security. [21] contains a description of a variety of approaches to multilevel security currently used in

secure object-oriented databases.

2.3 Federated databases

A federated database is a distributed database that "consists of component DBSs [database systems] that are autonomous yet participate in a federation to allow partial and controlled sharing of their data. Association autonomy implies that the component DBSs have control over the data they manage. They cooperate to allow different degrees of integration" [24. p189].

SeFD will assume that the component databases have compatible security systems and that the component databases are capable (and willing) to exchange the security information required for the operation of SeFD.

We assume that the federated database is homogeneous, not globally controlled and that no federal schema exists. See [24] for a discussion of these and other issues that exist for federated databases. The overview given by [22, 66–89] gives a clear positioning of federated databases amongst the possible alternatives for distributed databases. An introduction to distributed databases can also be found in [6].

We will assume that the local autonomy of each site implies that the site has definite rights over the information stored at that site. When we say that a site 'owns' local information, we will refer to these rights.

It is possible that ownership of information can be transferred; however, relocation of information does not necessarily imply transfer of ownership. This will be dealt with in detail later.

Also note that a federated database must "be able to grow incrementally and to operate continuously, with new sites joining to existing ones, without existing sites to agree with joining sites on global data structures or definitions" [4, p323]. This requirement will be taken into account when SeFD is described.

3 Security requirements of a federated database

As stated earlier, local autonomy is the distinguising characteristic of federated databases. And the site's ability to control access to its information is a fundamental aspect of local autonomy.

In the first instance, local autonomy implies local classification of local information. Similarly, one can argue that local autonomy also implies that the site should be able to determine the clearance of subjects directly associated with that site.

These two requirements have two implications affecting the site's participation in the federation. Firstly, the site's decision about the classification of its information should be respected throughout the federation. This implies that no member of the federation should disclose information to a party that the owner would not have disclosed it to. Secondly, a site may not be equally willing to share its information with all other sites in the federation. This may be because a particular site does not agree with the subject clearance assignment policy of another site (and therefore be unwilling to share, say, *top secret* information with that site because this site does not trust some subjects of that site who have been assigned *top secret* clearances by that site). The site may also be unwilling to share

sensitive information with that site because it has evidence (or suspicion) that the other site does disclose the information to unacceptable parties. The reason why some information is not to be disclosed to a particular site may also be the different roles different parties play in the federation. To illustrate, consider a federation of commercial databases consisting of bank and retail databases. The banks may be willing to share information with one another that they are not willing to share with retailers.

We will use the phrase *site trustedness of site A from the viewpoint of site B* to refer to the maximum sensitivity of information that site B is willing to share with site A. Often we will abbreviate this to the *trustedness of site A* when the identity of the owner site is obvious.

This means that for any site S and any sensitivity level L a set of trusted sites can be computed, that is sites where site S is willing to send information with sensitivity level L to. The notation $T(S, L)$ will be used to indicate such a set, and it will be referred to as the *trusted site set of S at level L*.

Secure interoperation always raises the issues of understanding and enforcement: firstly, how does the global system ensure that all component systems understand the security policy the same and, secondly, what guarantees can be provided that the other systems will properly enforce restrictions? In our case understanding at the technical level is trivially solved because the same security systems are used. Understanding at a higher level can be a problem: one site can use different criteria for classifying data or users than another site. The solution that we are proposing in this case is that data (at a given level) is simply not sent (directly or indirectly) to a site before the system security officer at the sending site is convinced that the receiving side has an acceptable security policy for data at the concerned level. Enforcement is ensured because we assume that the various sites use the same software. If this was not the case, one site will again need to convince the other site that it does enforce security properly before the other site will send information to this site.

This paper is therefore based on the following two fundamental security requirements of a federated database (in addition to the normal 'centralised' requirements that apply at each site):

1. Federation wide 'respect' for the owner's limitations on the treatment of its data; and
2. The ability of a site to limit the sensitivity of information owned by it to be sent to any particular site in the federation.

4 Proposed model

The next section gives some assumptions made about security in SeFD, in addition to the security requirements identified earlier for federated databases. Next *ownership* of data and the related concept of *trusteeship* are considered. This is followed by descriptions of cases where entities are relocated or instantiated (for example temporary object relocation, object emigration and object replication). This is followed by a treatment of changes in the federation.

4.1 Security assumptions

SeFD assumes that the individual sites are multilevel secure databases themselves. The security provided by SeFD coincide with that provided by the component databases, except for the additional requirements described in section 3.

Using the taxonomy described in [21], we make the following security assumptions about SeFD:

X1.1 The labels used to classify subjects and entities are partially ordered; a method can read from a variable if the clearance of the subject dominates ($\geq$) the sensitivity of the variable; a method can write to a variable if the clearance of the subject is dominated by the sensitivity of the variable; a method can be activated if the clearance of the subject dominates the sensitivity of the method.

X1.2 SeFD uses existence protection; that is, the fact that an entity exists is considered as sensitive as the information represented by (or contained by) the entity.

X2.1 Classes and objects can be protected (labelled), as well as their methods and instance variables.

X2.2 When an object is instantiated, it is labelled according to rules specified in its class; it can be relabelled by the system security officer at the site where the object resides.

X2.3 No additional restrictions apply except those restrictions that apply to all existence protected models identified in [21]: amongst others, the sensitivity of methods and instance variables dominates that of their object; instances are at least as sensitive as their classes and subclasses as sensitive as their superclasses.

X3.1 The authorisation of a message is determined by the clearance of its primary accessor. The authorisation can be reduced by the sites that participate in a request as detailed in subsequent sections.

X3.2 Message sensitivity is determined by the 'normal' rules as described in [21]: the sensitivity of a message is increased whenever it accesses (reads) a value more sensitive than the current sensitivity of the message. The sensitivity of a value is determined by the sensitivity of the object and variable that contains the value or the sensitivity of the method that returns the value.

X3.3 When a message cannot update the intended entity because the contents of the message is too sensitive, the request will be rejected.

Parameters X2.3, X3.2 and X3.3 do not have a significant influence on SeFD. The other parameters will be used in the description of SeFD.

4.2 Ownership

This section deals with the ownership rights that a site has over data stored at that site. Note that ownership as used in this case should not be confused with ownership in a model for discretionary security: In discretionary security an owner is a subject that has the discretionary power to share entities owned by it with other subjects (see [23, pp285–286] for example). In the current model *owner* refers to the *site* that has the authority to decide with which other *sites* the entity may be shared.

A fundamental property of mandatory security is that an entity should be as protected wherever it might be moved to in a system as it has been in its original location [21]. In an object-oriented database objects are continually sent (as parameters) with messages. (Remember we consider every data item in an object-oriented system to be an object.) In a distributed database, these objects will often be sent across site borders. Another possibility for objects to be moved across site borders is object relocation, either temporarily for query optimisation purposes, or permanently when the reason for placing information at a specific site changes. A last possibility for such moves across site borders is object replication at more than one site, in most cases for efficiency or reliability. Each of these three possibilities will be discussed shortly.

Before discussing the individual possibilities for such object movement, the principle to be used in these cases should be considered. If one agrees that, under normal circumstances, the object should be as protected in its new location as it has been in its original, it means that the classification label of the object (including labels associated with any facets of the object, such as methods and instance variables) should be transferred with the object. However, this is not adequate. If the site where this object resided originally has not been willing to share its contents with a site X, the new site where it resides now should also not share its contents with site X. Stated in terms of ownership: Any other site should treat an entity only according to the wishes of its owner, by only allowing subjects access to the entity if the owner would have allowed it and only sharing the entity with other sites if the owner is willing to share it.

In some cases ownership may be transferred to a new site; this may be the case when an object relocates permanently to another site. However, for temporary 'visits' to other sites, such as when a message includes an entity as a parameter, ownership will not change.

The following subsections discuss the various possibilities for objects that are not currently located at their owner sites.

4.3 Trusteeship

Since the distributed database operates by sending messages between the participating sites, it will often happen that the sites contain information owned by another site. Unless ownership changes together with the transferring of the information, the receiving site can only use the information in ways acceptable to the owner – in other words the receiving site acts as trustee for any information received in this way. This section considers the case where such information is received as part of a message sent to the site; subsequent sections deal with object relocation and replication.

SeFD assumes that information received as part of a message never implies a transfer of ownership to the receiving site. The receiving site is thus restricted when using such information. Some of the situations can be dealt with quickly and those will be addressed first.

Suppose that an object residing at site A wants to send a message to an object residing at site B. However, assume that sites A and B are not connected, but both are connected to a third site C. Also assume that the sensitivity of the message to be sent is such that site A is willing to send it to site B, but does not trust site C enough to accept a message of this sensitivity. SeFD assumes that the underlying communication system is trusted and that the communication system is able to route information

via a not-so-trusted site in a secure way. This can be done by, for example, encrypting the message when it is transmitted at the sending site such that it can only be decrypted at its intended destination. A site that merely routes a message, therefore does not 'see' any contents of the message.

A related, but more complex problem concerns the case where a message is not simply routed via an intermediate site but sent to an object residing at that site and where that object then sends a message to a third site. Suppose, for example, that an object at site A wants to send a *top secret* message to an object at site B. Suppose further that site A does not want to send *top secret* information to site C. However, suppose that site B does trust site C enough to send *top secret* information to it. What are the implications if the target object at site B now sends a message to an object at site C, containing the *top secret* information originally sent by the object at site A? The conservative (but safe) approach usually followed by security models is to assume that a message that is sent following receipt of another message is at least as sensitive as the received message. This means if the received message was not supposed to be sent to a site C, no subsequent messages (in the current request) can be sent to site C.

In order to discuss possible solutions, we introduce the term *message trusted site set* to refer to the set of sites that can still participate in the request. The message trusted site set (logically or physically) accompanies the message. This set initially consists of all sites. Whenever a site S contributes information, all sites that the contributed information should not be sent to, are removed from the set. Whenever a message has to be sent to another site, the sending site will check the set to determine whether the message can indeed be sent to that site. The message trusted site set will also be referred to as the *message set* for the sake of brevity.

The message set need not be represented physically: it can be computed. However, SeFD does include the message trusted site set with the message. A possible implementation strategy will be discussed shortly.

This solution to determine the message trusted site set lies midway between two other possibilities:

- On the one side the sending side can contain enough information about other sites that makes it unnecessary to consider the message trusted site set dynamically, because a request will never be initiated that sends information to an unacceptable site.

  This solution has a number of drawbacks. Firstly, implementation details (including objects used to implement other objects) are likely to be confidential information in a federated database and not easily shared with all other sites. Further, it is suggested that one site is not allowed to inform a second site about entities available on a third site. Thirdly, it is against the spirit of the encapsulation principle to make use of such encapsulated information at all; in particular, if the implementation of an object is changed it may cause a ripple effect throughout the security system of the federated database. Lastly, the information that needs to be transmitted before the trustedness of all objects (or methods) can be determined seems to be prohibitively high.

- On the other side the message trusted set may not be included with the message, but rather computed when required.

  A drawback of this solution is the high communication overhead for all the anticipated approval requests.

Note that the option followed by SeFD to include the message trusted site set with the message need not incur excessive overhead: a simple solution (that will be revised later) is to include a bit string with every message, with one bit per site[1]. A one can then indicate that that site may still be involved in processing, while a zero may indicate the contrary. When a message originates from a primary user (for example a human operator) all bits are set to one. A site can then remove any site from the potential contributors by just setting the corresponding bit to zero for any message that it sends. No other site is allowed to change any bit to a one.

The message handler of any site now needs to do the following when it receives a message:

- Start a method activation for the indicated method.
- If data is accessed at any point by the method activation, determine the sensitivity of the data.
- Determine the trusted site set of the site where the message is executing for the sensitivity level of the data just accessed (as a bit string) and logically AND this set with the bit string accompanying the message to get a new message trusted site set.
- If the sensitivity level of the message does not dominate the sensitivity of the data just accessed, set the sensitivity of the message to the least upper bound of its current sensitivity and that of the accessed data.
- If the method activation attempts to write to an entity and the sensitivity of the entity does not dominate the sensitivity of the message, abort the message.

The message handler descibed above has to be a trusted process on the site where it executes.

We therefore conclude that

- A secure communication system has to be used to link various sites (implying that only the sending and receiving sites will be able to access the message contents); and
- Any message must carry with it (logically or physically) a list of sites that may be involved in handling ('executing') the message or receive data obtained as a result of this message.

4.4 Temporary object relocation

When an object moves temporarily to another site it will probably happen in order to optimise some query. Obviously, this does not mean that its 'security' should change—the restrictions imposed by the owner site still apply. It is therefore necessary for the new location to take such restrictions into account.

This may be implemented by attaching the owner site's trusted site set for every sensitivity level to the relocated object. The sending site can then take this into account whenever the object takes part in an exchange of messages. However, the trusted site sets are likely to be confidential information not readily shared with other members of the federation.

[1] As one referee pointed out, a federation of 8000 sites implies an overhead of approximately 1 kilobyte per message—again an indication that security can be costly.

An alternative implementation that is more likely to be acceptable is to expect the objects to protect themselves while residing at the other site: the security relevant information (current message sensitivity level and message trusted site set) now forms inherent parameters whenever any method is accessed. The method then updates the message set of acceptable locations appropriately. The information necessary to update this set can be contained as part (data) of the relocated object or can be obtained by this object from its owner site. A combination approach is also possible where the object may contain less sensitive sets internally, but requests a server at its owner site to update the message set for more sensitive site sets. This choice of which technique will be used for a particular object is left to the owner site of the concerned object. The owner site may also take additional steps to protect access sets included in the object, such as encrypting these sets. The object can further be protected by (physically) removing facets that should not leave the site at all.

We therefore recommend that the responsibility to update the message trusted site set is encapsulated in each object and the implementation details of this action be left to the owner site of the object.

Note that there may be sites to which an object cannot move; see section 4.7 (*Object instantiation*) for details.

4.5 Object emigration

Permanent relocation of an object will be referred to as *emigration*. When an object emigrates to another site ownership is likely to change to the new site. If ownership does not change, emigration can be handled exactly like temporary relocation.

If ownership does change, the trusted site sets of the old and new owners can be compared; if the new owner has the same (or a stricter) view of other sites than the old site, the object can be relocated without any problems. However, this process cannot be automated because the sites cannot access one another's trusted site sets. Further, if the new site is willing to send information to some site not acceptable to the old site, the emigration cannot occur.

In all cases manual intervention is the appropriate action: the concerned system security officers should decide whether the emigration can occur and, if it does occur, the entity becomes the property of the new site and is not owned by the old site in any way anymore.

Note again that there may be sites to which an object cannot emigrate; see section 4.7 (*Object instantiation*) for details.

4.6 Object replication

An object is replicated if copies of a (logically) single object occurs at more than one site (see for example [5, p624]). Replication presents a number of interesting problems. in particular propagating an update from any one copy to all copies (see for example [6, p293]).

Consider ownership of such a replicated object. For simplicity, SeFD only allows single ownership. This matches the idea of a primary copy of the replicated object (see for example [5, p630]); the owner of the primary copy then owns the replicated object (including all copies). The security of copies of the replicated object are then treated exactly like objects that have temporarily been relocated.

Note that there may be sites where a specific object cannot be replicated; see section 4.7 (*Object instantiation*) for details.

4.7 Object instantiation

An object carries with it information about its class. Since SeFD uses existence protection, information such as which methods a particular object support (and hence which methods are available from its class) are considered protected information. The question that needs attention is whether an object can be instantiated at any site, or, for example, only at the site containing its class.

Since the structure of the class is only to be shared with those sites trusted enough by the site owning the class, an *object* can clearly only be instantiated at sites acceptable to the owner of the *class*. The set of such sites depends on the classification level (sensitivity) of the class. Further, once instantiated, the object cannot be relocated to a site where it could not have been instantiated in the first place.

SeFD uses the following mechanism to ensure proper instantiation and relocation of objects: To instantiate a new object, a message is sent to the class of the new object. The method carries with it the site where the instantiated object is to reside. (As a default, the site where the instantiation request originated will be the site containing the new object). The class then verifies (with its owning site) whether the object can be instantiated at the requested site, and instantiates it if acceptable; if not, the request is denied. In addition, all objects contain a method that determines whether any new site is an acceptable location for the object; before the object is relocated (moved or replicated) the site currently containing the object will activate this message to verify that the planned new site is indeed acceptable. This method can either contain the list of acceptable destinations, or can check with the site containing its class.

Subclass creation has a similar problem: If the subclass is instantiated at site Y while its superclass is owned by site X, obviously any information about the superclass inherited by the subclass should be treated according to the wishes of site X. The conservative approach followed by SeFD is to allow the subclass (or its instances) to be accessed only by sites acceptable to both site X and Y. This is accomplished by requiring that an instance of any subclass has the option to deny a request to relocate it to another site. If it does not deny the request, the decision is made by the class, exactly like the decision would have been made for a request to relocate an instance of the class.

All classes therefore include an *acceptable site* method that is invoked when an object is moved. This same method is used by the *create* method of the class. Further, this method will always request permission from the corresponding method of its superclass before granting the requested approval.

4.8 Relocation of classes

SeFD assumes that classes are objects themselves (the view held by Smalltalk [9]). Therefore, for a class to be relocated (temporarily or for emigration or replication) the same restrictions apply that apply to object relocation.

However, in addition to the 'object' restrictions, additional restrictions apply: Every class C that is a subclass of another class S carries with it information about the class S. Such a subclass will carry with it the site contraints of its superclass S, just like any instance of S will carry with it such constraints. These constraints will be checked in addition to the 'object' restrictions mentioned earlier.

4.9 Changes in the federation

It has been mentioned earlier that federations change: Sites join and leave the federation continually. Further, a site may change its view of the trustedness of other sites; in this case its trusted site sets will change.

The first form of change (sites joining and leaving) can be a problem if bit strings are used to represent trusted site sets. In this case every site has an inherent number associated: the position that it occupies in the bit strings. New sites will be assigned a position in the bit string as part of the negotiation protocol to join the federation (negotiation protocols are used extensively to cooperate activities in a federated database—see [24, pp221–222] for an introduction). All members will (eventually) be informed of the existence of the new member. However, until a site is informed of the existence of the new member, it will consider the new member untrusted for all sensitivity levels and not send any messages to it. When a site is informed of the existence of the new site, the local system security officer is informed accordingly, after which the trustedness of this site is determined and the trusted site sets modified. Objects that contain copies of the trusted site sets also have to be updated. This will be dealt with in the following paragraph. When a site leaves the federation, the trusted site sets will similarly have to be updated. Note that an adequate time has to elapse before the corresponding bit position is used again, because all sites must be able to remove the site that has left before the bit position can be reused.

From the second form of change (ie change of trusted site sets) the problem arises because of trusted site sets that are part of objects—especially objects that are not (currently) located at their owning site. This is relatively easily solved by requiring each site to maintain a list of objects that it owns, residing at other sites. Such objects will then contain a method to update their stored trusted site sets that can be activated by the owning site. The same method will be used to update the trusted site sets in the event of a new site joining or leaving the federation. When trusted site sets change, the owning site will also be able to identify its objects that reside at sites that are no longer trusted and relocate those objects. Note that a request from an owner to relocate the object to the owner has to be accepted by any site—even if it is not normally willing to send information at the concerned sensitivity level to the owner site.

5 Conclusion

Trusteeship is a central issue in a federated database; often members of the federation handle information on behalf of other members. Where this occurs, this member acts as a trustee for the information owned by the other member.

SeFD uses a mechanism where such wishes are obtained from the concerned object. The owner of the object can decide on the way an object 'knows' the wishes of its owner—by including such 'wishes' in the object and/or instructing the object to obtain such 'wishes' from the owner. Objects modify the message trusted site set to protect information released by the object. The object is also able to deny a request to relocate it to another site based on the constraints imposed by its owner.

Since trust plays a central role in a federated database, it is necessary to limit the actions of unworthy trustees. SeFD allows this to be done on a site by site basis: each site has the discretionary power to limit the sensitvity of any information that may (eventually) flow from this site to any other specific site in the federation.

The following questions were identified during the design of SeFD but not addressed by the current paper:

- SeFD assumes that the sites of the federated database are homogeneous. no provision was made for heterogeneous sites. As one example, assume that the one site uses a fully ordered set of classification labels, while the other uses a partially ordered set. It is clear that the one that uses the partially ordered set should accept data from the other without too much of a problem. However, the converse does not hold because there is no obvious, universally applicable way that the partially ordered labels can be translated to fully ordered labels.
- It may be possible that a request cannot continue because a message cannot be sent to a target object because it resides on a site that is (no longer) in the message trusted site set. In this case it may be possible to relocate the object to a site that does still appear in the message trusted site set and so allow processing to continue; this option needs to be investigated
- In SeFD different aspects of an object can be owned by different sites. Typically structural information of an object is owned by the owner of the object's class (and its superclasses), while the content is owned by the site where the object was instantiated or emigrated to. However, SeFD does not make provision for different aspects of an object to be owned by different sites.
- SeFD assumes that the object is the basic entity that will be relocated or replicated—no provision was made for fragmentation.

These points remain interesting research questions that will receive attention in future.

References

[1] M Atkinson *et al* "Object-oriented Database System Manifesto", *Proceedings of the first international Conference on Deductive and Object-oriented Databases*. Kyoto, Japan. December 1989

[2] T Atwood, "The Object-oriented Database System Manifesto: A Consensus from Academia", *Hotline on Object-oriented Technology*, **1**. 3, 6–9, January 1990

[3] JA Bull, L Gong and KR Sollins, "Towards Security in an Open Systems Federation", *Proceedings of European Symposium on Research in Computer Security*. Lecture Notes in Computer Science Volume 648, 3–20, Springer-Verlag, 1992

[4] S Ceri and G Pelagatti, *Distributed Databases*, McGraw-Hill, 1985

[5] CJ Date, *An Introduction to Database Systems Volume 1, 5th ed.*, Addison-Wesley, 1990

[6] CJ Date, *An Introduction to Database Systems Volume 2*, Addison-Wesley, 1985

[7] DE Denning, "Database Security", 1–22 in [30], 1988

[8] CG Gable and WJ Caelli (eds), *IT Security: The need for International Cooperation*, North-Holland, 1992

[9] A Goldberg and D Robson, *Smalltalk 80: The Language and its Implementation*, Addison-Wesley, 1983

[10] L Gong and X Qian, "The complexity and Composability of Secure Interoperation", *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, 190–200, Oakland, California, 1994

[11] S Jajodia and B Kogan, "Integrating an Object-oriented Data Model with Multilevel Security", *IEEE Symposium on Research in Security and Privacy*, 76–85, 1990

[12] TF Keefe, WT Tsai and MB Thuraisingham, "SODA: A Secure Object-oriented Database System", *Computers & Security*, **8**, 1989, 517–533

[13] TF Keefe and WT Tsai, "Prototyping the SODA Security Model", 211–235 in [26], 1990

[14] W Kim, "Object-oriented database systems: strengths and weaknesses", *Journal of Object-oriented Programming*, **4**, 4, 1991, 21–29

[15] TF Lunt, "Multilevel Security for Object-Oriented Database Systems", 199–209 in [26], 1990

[16] TF Lunt, DE Denning, RR Schell, M Heckman and WR Shockley, "The SeaView Security Model", *IEEE Transactions on Software Engineering*, **16**, 6, 1990, 1247–1257

[17] TF Lunt and PK Boucher, "The SeaView Prototype: Project Summary", *Proceedings of the 17th National Computer Security Conference*, Maryland, October 1994

[18] J McHugh and BM Thuraisingham, "Multilevel Security Issues in Distributed Database Management Systems", *Computers & Security*, **7**, 1988, 387–396

[19] MS Olivier and SH von Solms, "Building a Secure Database Using Self-protecting Objects", *Computers & Security*, **11**, 3, 1992, 259–271

[20] MS Olivier and SH von Solms, "DISCO: A Discretionary Security Model for Object-oriented Databases", 345–357 in [8], 1992

[21] MS Olivier and SH von Solms, "A Taxonomy for Secure Object-oriented Databases", *ACM Transactions on Database Systems*, **19**, 1, 1994, 3–46

[22] MT Özsu and P Valduriez, *Principles of Distributed Database Systems*, Prentice-Hall, 1991

[23] CP Pfleeger, *Security in Computing*, Prentice-Hall, 1989

[24] AP Sheth and JA Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, **22**, 3, 1990, 183–236

[25] B Shriver and P Wegner (eds), *Research Directions in Object-Oriented Programming*, MIT Press, 1988

[26] DL Spooner and C Landwehr (eds), *Database Security III: Status and Prospects*, North-Holland, 1990

[27] BM Thuraisingham, "Mandatory Security in Object-Oriented Database Systems", *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, ACM, October 1989, 203–210

[28] BM Thuraisingham, "Multilevel Security Issues in Distributed Database Management Systems II", *Computers & Security*, **10**, 1991, 727–747

[29] BM Thuraisingham and HH Rubinowitz, "Multilevel Security Issues in Distributed Database Management Systems III", *Computers & Security*, **11**, 1992, 661–674

[30] JF Traub *et al* (eds), *Annual Review of Computer Science Volume 3*, Annual Reviews inc, 1988

[31] V Varadharajan and S Black, "Multilevel Security in a Distributed Object-Oriented System", *Computers & Security*, **10**, 1991, 51–67

[32] P Wegner, "The Object-Oriented Classification Paradigm", 479–560 in [25], 1988

[33] P Wegner, "Concepts and Paradigms of Object-Oriented Programming", *OOPS Messenger*, **1**, 1, 1990, 7–87

A New Authorization Model for Object-Oriented Databases

Elisa Bertino Fabio Origgi Pierangela Samarati

Dipartimento di Scienze dell'Informazione
Università di Milano
Milano, Italy

Abstract

In this paper we present an authorization model for the protection of object oriented databases. The model supports the concepts of explicit/implicit, positive/negative, and strong/weak authorization. The model is an evolution of the ORION authorization model but differs from this in many respects. In particular, the semantics of subject groups and of negative authorizations applied to sets of objects is different. As a consequence also the implication rules for the derivation of authorizations are different. In the paper we present our authorization model, illustrate the implication rules supported by our model for the derivation of authorizations in object-oriented systems, and define overriding rules among authorizations.

1 Introduction

Object-oriented database management systems (OODBMS) represent today the most important research and development direction in the area of database systems. Indeed, in addition to OODBMS directly developed from the object-oriented paradigm, a number of systems, like relational DBMS (RDBMS) and deductive DBMS are currently being extended with object-oriented capabilities. The most notable example is SQL-3 [11], the new standard, currently under definition, for the SQL language.

Most attention in the development of data management systems with object-oriented features has, however, been given to traditional database issues such as data modeling, query languages, query processing, and schema management. By contrast, less attention has been given to the problem of access control. Therefore, most of those systems still lack adequate authorization models. The reason is that the various authorization models, developed for access controls in operating systems and in data base management systems, cannot be directly applied to OODBMS. Indeed, several concepts of the object-oriented data model, such as inheritance, versions, and composite objects, introduce new protection requirements which the traditional authorization models do not address. Another source of complexity is that the applications intended as target for OODBMS, like CSCW, office automation, CAD, have additional protection requirements and policies that need adequate support from authorization mechanisms. Temporal authorizations represent an example of such a requirement [2].

Work in the area of authorization models for object-oriented databases is still in a preliminary stage [5]. Very few OODBMS, namely Orion [13] and Iris [1], provide authorization models comparable to the models provided by current Relational DBMS. Other OODBMS either do not provide any authorization mechanism at all, or provide very low level capabilities. The GemStone system, for example, only allows authorizations to be associated with segments, where a segment is the storage unit for objects [4].

The goal of the work, presented in this paper, is to define a new comprehensive authorization model for OODBMS, characterized by a formal and sound basis. Indeed, the complexity of authorization models for OODBMS requires formal foundations for accounting all aspects of these models and as a basis for a correct implementation. Our model has been defined as an evolution of the Orion authorization model. Our model, as the Orion model, provides the notions of explicit/implicit authorization, positive/negative authorization, and strong/weak authorization. However, it has a number of important differences with respect to the Orion model.

First, we take the approach of increasing the authorization types of the model and reducing the number of authorization objects. The Orion model is based on only four authorization types, whereas the number of authorization objects is quite high. The main drawback of the Orion model is that authorization objects are introduced that do not always correspond to real database objects. Moreover, the same authorization type applied to different object types may have different semantic meanings, thus making the semantics hard to understand in some cases. The model presented here is therefore more natural and can be easily mapped onto a user language. Moreover, our model allows a more detailed modeling of implications among authorization types.

Second, the Orion model uses a notion of *user role* that does not clearly correspond to either the notion of group nor to the notion of role, as currently supported by RDBMS or defined in literature proposals. Our model is based on the notions of *user* and *group* as currently supported in RDBMS. Moreover, we plan to add the notion of user role in our model, with, however, a clear semantic difference with respect to the notion of group [1].

A third difference is related to semantics of negative authorizations. Both our model and the Orion model support negative authorization as a way to support exceptions with respect to authorizations collectively granted to a set of subjects, or on a set of objects. However, the ORION model the semantics of negative authorizations on a set of objects is not clearly defined. In particular, a negative authorization on a set of objects does not denote a negative authorization on each object of the set. Therefore, it is not possible to specify negative authorizations on a set of objects to be intended as applicable to all objects of the sets. For instance, the only way to give a user the negative authorization for the read privilege on all instances of a class is give the user the negation to read the definition of the class. However, this negation is more than what we would have like to do (it does not allow to read the definition of the class) and it does not admit exceptions [3].

Finally, other differences that will not be discussed in details in the present paper include a more sophisticated management for authorizations on versions and composite objects, and capabilities for decentralized authorization administration [3].

With respect to authorization models described in other papers (e.g. [9, 1, 8])

[1] Note that recent versions of RDBMS support both the notions of group and role.

our model differs for several aspects. First, the model defined by Dittrich et Al. [9] does not include many of the notions that the Orion model and our model include, namely positive/negative authorization, and strong/weak authorization. Indeed, the model by Dittrich et Al. only provides the notion of explicit/implicit authorizations; implicit authorizations in this model are derived through deductive rules defined by the users. No consistency criteria are, however, defined by Dittrich et Al. for those deductive rules. The model defined for the Iris system [1][2] is based on the approach of considering methods as the authorization access types. Therefore, in that model users are authorized to invoke methods on objects. The Iris model, however, does not have a formal basis, and several questions, especially concerning authorization administration, are left open. Moreover, the Iris model does not provide the same flexibility of our model, since it does not provide all the various types of authorizations, like positive/negative authorization and so on. The model defined by Bruggemann [8] differs with respect to our model in a number of aspects. First, in that model no complete account is given for all modeling constructs of an object-oriented data model, in that this model does not consider composite objects, and versions. Moreover, no specification is provided of the access types supported in the model, whereas our model provides a complete account under this aspect. Finally, that model handles exceptions based on an explicit ordering given by the users. However, it is not clear how this approach would work in a decentralized authorization environment. By contrast, our model supports multiple-level exceptions based on the hierarchical composition of authorization objects, and of user groups. Therefore, our model does not require explicit user-defined priorities among exceptions.

The remainder of this paper is organized as follows. Section 2 describes the reference object-oriented data model which will be used throughout in paper. Section 3 illustrates our authorization model. Section 4 presents the implication rules for the derivation of authorizations. Section 5 defines the authorization state. Section 6 illustrates how the access control works. Finally, Section 7 presents the conclusions.

2 A reference object-oriented data model

In this section, we summarize the main features of object-oriented data models by a reference model which will be used in the rest of the paper for the discussion.

Each real-world entity is modeled as an object. A set of attributes is associated to each object. An attribute of an object may take on a single value or a set of values. Each object is associated with a unique identifier (OID) which is fixed for the whole life of the object. The identity of an object has an existence independent of the values of the object attributes.

All objects which share the same set of attributes are grouped together in a higher level object called a *class*. Each object belongs to (is an *instance of*) only one class. A *primitive* class is a class with no attributes (e.g., integer, string, or boolean). The value of an attribute of an object belongs to some class. This class is called the *domain* of the attribute. The domain of an attribute may be any class, including a primitive class. If an attribute of a class C has a class C' as a domain, an *aggregation relationship* is established between the classes C and C'. According to this relationship, the set of

[2]Note that this model has not been actually implemented in the Iris prototype. Therefore, the model described in the paper is still at preliminary stage.

classes in the schema is organized in an *aggregation hierarchy*.

Users can derive a new class from an existing class. The new class, called a *subclass* of the existing class, inherits all the attributes of the existing class, called the *superclass* of the new class. The instances of the subclass are *members* of all its superclasses. Users may specify additional attributes for the subclass. A class may have more than one subclass (*multiple inheritance*). According to the subclass/superclass relationship, the classes form a rooted directed acyclic graph, called *inheritance hierarchy*.

Objects are accessed by *system-defined* methods that allow to read and write object attributes, read and modify the class definitions, create and delete instances and classes, and so on.

3 The authorization model

In this section we illustrate our authorization model.

3.1 Elements of the model

3.1.1 Subjects

Authorization subjects can be either users or groups. In the following, sets S, U, and GS denote the sets of subjects, users, and groups respectively, where $S = U \cup GS$. A group is defined as a set of other subjects (users or groups). Groups are not necessarily disjoint, i.e., a same subject may belong to more groups. The membership relationship between subjects is represented by a graph, called *subject graph*, as follows. Each subject is represented by a node. An arc directed from subject s_i (group) to subject s_j (user or group) indicates that subject s_j directly belongs to group s_i. An example of subject graph is illustrated in Figure 1.

Given two subjects $s_j \in S, s_i \in GS$, notation $s_j <_1 s_i$ (or simply, $s_j < s_i$) indicates that there exists an arc from s_i to s_j in the subject graph, i.e., s_j directly belongs to group s_i. Notation $s_j <_n s_i$ indicates that there exist $s_{k_0} \in S, s_{k_1}, \ldots, s_{k_n} \in GS$ such that $s_{k_0} = s_j, s_{k_n} = s_i$ and $s_{k_0} < s_{k_1} < \cdots < s_{k_n}$. Notation $s_j <_0 s_i$ indicates equality, i.e., is equivalent to $s_j = s_i$.

If $s_j <_n s_i$ with $n > 1$ we say that s_j indirectly belongs to group s_i. A subject s_j may belong to a group s_i through several paths in the subject graph. In other words, the relation $s_j <_n s_i$ may be valid for different values of n. For instance, with reference to the subject graph illustrated in Figure 1, Bob $< G_4$, Bob $< G_2$, Bob $<_2 G_2$, Bob $<_2 G_1$, Bob $<_3 G_1$.

3.1.2 Objects

The set of authorization objects, denoted by O, is composed of databases, classes of the databases, and instances of the classes, i.e., $O = Database \cup Class \cup Instance$. We do not consider attributes of instances as objects of authorizations. However, authorizations for specific attributes can be specified on instances.

Objects are organized into an *object granularity hierarchy*: a class is composed of a set of instances, a database is composed of a set of classes. The system in turn consists of a set of databases. An example of object granularity hierarchy is illustrated in Figure 2.

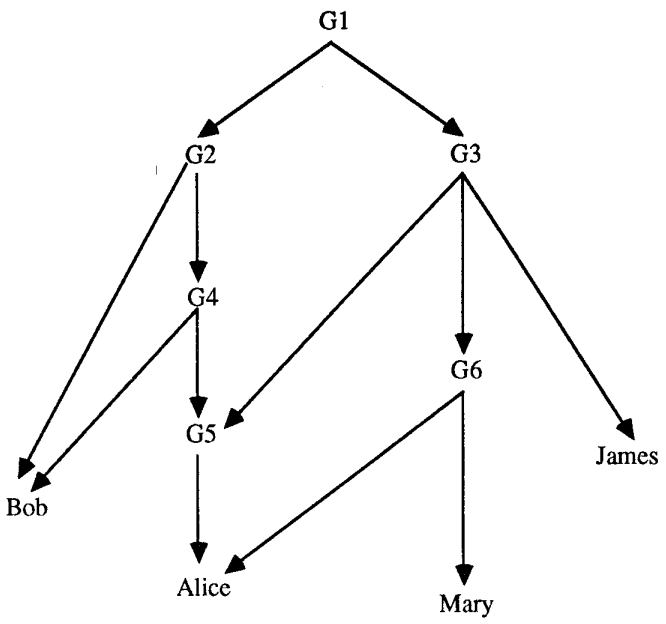


Figure 1: An example of subject graph

In the following, given two objects $o_i, o_j \in O$, notation $o_j <_1 o_i$ (or simply, $o_j < o_i$) indicates that o_j is a direct descendant of o_i in the object granularity hierarchy, i.e., o_j directly belongs to the set of objects represented by o_i. Notation $o_j <_n o_i$ indicates that there exist $o_{k_0}, o_{k_1}, \ldots, o_{k_n} \in O$ such that $o_{k_0} = o_j, o_{k_n} = o_i$ and $o_{k_0} < o_{k_1} < \cdots < o_{k_n}$. Notation $o_j <_0 o_i$ indicates equality, i.e., is equivalent to $o_j = o_i$.

For instance, with reference to the object granularity hierarchy of Figure 2, Empl < Employees < Administration.

Classes are organized into inheritance hierarchies on the basis of the subclass/superclass relationship. In the following, given two classes o, o' notation $o' \prec_1 o$ (or simply, $o' \prec o$) indicates that o' is a direct subclass of o. In multiple inheritance, given $o_1, o_2, \ldots, o_n \in \mathit{Class}$, notation $o' \prec \{o_1, o_2, \ldots, o_n\}$ indicates that o' is a direct subclass of $o_1, o_2, \ldots, o_n$. Notation $o' \prec_n o$ indicates that there exist $o_{k_0}, o_{k_1}, \ldots, o_{k_n} \in \mathit{Class}$ such that $o_{k_0} = o', o_{k_n} = o$ and $o_{k_0} \prec o_{k_1} \prec \cdots \prec o_{k_n}$. Again, notation $o' \prec_0 o$ indicates identity, i.e., is equivalent to $o' = o$.

3.1.3 Access modes

The set of access modes, denoted by M, consists of privileges that users can exercise on the objects. The access modes applicable to an object depend on the type of the object, e.g., database, class, or instance. In our model, the following access modes are considered:

1. $M(\mathit{database})$ = **Access modes applicable to databases:**
   (a) *read_def*: to read the definition of the database;

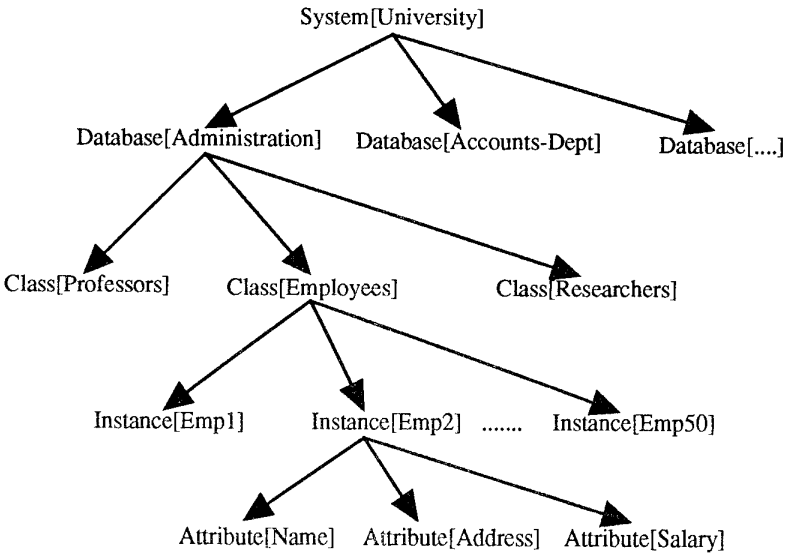


Figure 2: An example of object granularity hierarchy

(b) *read*: to read all objects within the database;
(c) *write*: to modify all objects within the database;
(d) *create*: to create a new class within the database.

2. $M(class)$ = **Access modes applicable to classes**:
   (a) *read_def*: to read the definition of the class;
   (b) *write_def*: to modify the definition of the class;
   (c) *delete_def*: to drop the class;
   (d) *read*: to read all instances of the class;
   (e) *write*: to modify all instances of the class;
   (f) *read*(A_k): to read attribute A_k for all instances of the class;
   (g) *write*(A_k): to modify attribute A_k for all instances of the class;
   (h) *create*: to create new instances of the class;
   (i) *delete*: to delete all instances of the class.
3. $M(instance)$ = **Access modes applicable to instances**:
   (a) *read*: to read all attributes of the instance;
   (b) *write*: to modify all attributes of the instance;
   (c) *read*(A_k): to read attribute A_k of the instance;
   (d) *write*(A_k): to modify attribute A_k of the instance;
   (e) *delete*: to delete the instance.

The *read_def* access mode on a class allows to read the definition of the class. The *read_def* access mode on a database allows to read the directory of the database, i.e., it allows to see the names of all classes within the database and the inheritance hierarchies

they form. Note, however, the *read_def* access mode on a database does not allow users to read the definition of the classes contained in the database.

The *delete_def* access mode, which allows to drop a class, can be executed only when the class has no instances.

The *write* privilege on a database is very powerful since it allows every access to the database and the objects contained in it. Therefore, it is intended for the use of the database administrator.

The privilege to read an attribute of an object whose value is the OID of another object allows to read the object identifier of the referenced object, but, in general, it does not allow to read the contents of the referenced objects.

In order to ensure that an access mode is properly applied, we define a function $t : O \rightarrow \{database, class, instance\}$ which associates with each object $o \in O$, its type $t(o)$. An access mode $m \in M$ is applicable to an object $o \in O$ if and only if $m \in M(t(o))$.

3.2 Authorizations

Our model provides both positive and negative authorizations. A *positive* authorization is used to specify that a subject may exercise an access mode on an object. A *negative* authorization is used to specify that a subject is denied an access mode on an object.

The possibility of specifying both positive and negative authorizations may introduce conflicts due to the simultaneous presence of two authorizations that differ only in the sign. In some cases, conflicts may be easily resolved by specifying overriding rules among authorizations. Our model gives the grantor of an authorization the possibility of specifying whether the authorization may be overridden by other authorizations. To support overriding, we distinguish between strong and weak authorizations. A *strong* authorization cannot be overridden, i.e., it does not admit exceptions. A *weak* authorization may be overridden, according to specified rules, by other strong or weak authorizations.

To formally represent authorizations, we introduce the following definitions.

Definition 1 (Authorization Space) The *authorization space* ASP is defined as:

$$ASP = S \times O \times M \times \{+, -\} \times \{st, wk\}$$

where $+$ indicates positive, $-$ indicates negative, st indicates strong, and wk indicates weak.

Definition 2 (Authorization) An *authorization* $a \in ASP$ is a 5-tuple (s, o, m, as, at) where:

$s \in S$ is the subject to whom the authorization is granted;

$o \in O$ is the object to which the authorization is referred;

$m \in M(t(o))$ is the access mode;

$as \in \{+, -\}$ indicates whether the authorization is referred to the access mode $(+)$ or its negation $(-)$;

$at \in \{st, wk\}$ indicates whether the authorization is strong (st) or weak (wk).

For instance, authorization $(s, o, m, +, st)$ states that subject s can exercise access mode m on object o, and this authorization cannot have exceptions. Authorization $(s, o, m, -, wk)$ states that subject s cannot exercise access mode m on object o, and this authorization can have exceptions.

Given an authorization a, notation $s(a)$, $o(a)$, $m(a)$, $as(a)$, $at(a)$ denotes the subject, the object, the access mode, the sign (positive or negative), and the type (strong or weak) of authorization a, respectively.

Authorizations specified by the users are called *explicit*. These authorizations are grouped into a strong and a weak authorization base as follows.

Definition 3 (Strong authorization base) A *strong authorization base* SAB $\subset ASP$ is a set of explicit authorizations with $at(a) = st$.

Definition 4 (Weak authorization base) A *weak authorization base* WAB $\subset ASP$ is a set of explicit authorizations with $at(a) = wk$.

The authorizations specified by the users are seen as a generating set of authorizations. Starting from these authorizations, other authorizations can be derived through the relationships among subjects, objects, and access modes. Derivation rules are given in Section 4.

We introduce an *implication relationship* between authorizations defined as follows. Given two authorizations $a, a' \in ASP$, a *implies* a', denoted by $a \rightarrow a'$, if a' is derived from authorization a through one of the implication rules of the model. Notation $a \rightarrow_n a'$ indicates that there exist $a_0, a_1, \ldots, a_n \in ASP$ such that $a_0 = a, a_n = a'$ and $a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n$. If $n = 0$ the relationship indicates equality, i.e., writing $a \rightarrow_0 a'$ is the same as writing $a = a'$.

The implication relationship preserves the authorization sign and the authorization type, i.e., given $a, a' \in ASP$, if $a \rightarrow_n a'$ then $as(a') = as(a), at(a') = at(a)$.

Authorizations derived by applying the implication rules are called *implicit*, as stated by the following definition.

Definition 5 (Implicit authorization) An *implicit* authorization is an authorization $a \in ASP$ such that $a \notin$ SAB $\cup$ WAB and $\exists a' \in$ SAB $\cup$ WAB such that $a' \rightarrow_n a, n \geq 1$.

Note that an authorization explicitly contained in an authorization base may also be derivable from other authorizations through the implication relationship. According to the previous definition we consider this authorization as explicit. In other words, we do not require the explicit set of authorizations to be minimal.

4 Implication rules

In this section we illustrate the implication rules of our model. We distinguish among the different domains along which we derive implicit authorizations. We analyze implication rules for subjects, for access modes, for objects, and along the class inheritance hierarchy.

Implication rules can be illustrated with a graph in which the nodes are the access modes. The set of nodes is partitioned into three disjoint subsets according to the

type of object to which the access modes are referred (i.e., databases, classes, and instances). The arcs represent the implication rules of our model. The black-colored arcs represent implications between positive authorizations, while the grey-colored arcs represent implications between negative authorizations. Each arc is labeled with the number of the corresponding implication rule.

4.1 Implication rules for subjects

The first implication rule refers to the propagation of positive and negative authorizations along the subject graph. The consideration of groups of subjects allows to grant privileges to set of users in an efficient way.

Rule 1 The authorization (negation) of a group for a privilege on an object implies the authorization (negation) for the privilege on the object for the direct members of the group. Formally, $\forall s_i, s_j \in S, \forall o \in O, \forall m \in M(t(o)), \forall as \in \{+,-\}, \forall at \in \{st, wk\}, s_j < s_i : (s_i, o, m, as, at) \rightarrow (s_j, o, m, as, at)$.

Rule 1 states that an authorization a for a subject (group) s propagates to all the subjects which *directly* belong to s. By applying this rule recursively we have that the authorization of a group propagates to all its members, both direct and indirect.

For instance, with reference to the subject graph illustrated in Figure 1, authorization (G_1,Employees,*read_def*,+, wk) implies authorization (G_2,Employees,*read_def*,+, wk) which in turn implies authorization (Bob,Employees,*read_def*,+, wk).

4.2 Implication rules for access modes

Implication rules for access modes are based on relationships among access modes which belong to the same subset of access modes (i.e., referred to the same type of object). Given an authorization (negation) for an access mode on an object, these rules allow an authorization (negation) to be derived for another access mode on the same object. Negative authorizations propagate, with respect to the corresponding positive authorizations, either in the same way or in the opposite way (logic negation).

In the following, notation *Setof_attr*(o) denotes the set of attributes of object o. The reason for the implications expressed by the rules is of immediate interpretation; we will include some explanation when needed. The graphical representation of these rules is given in Figure 3.

Rule 2 The privilege to modify an object implies the privilege to read the object. Formally, $\forall s \in S, \forall o \in O, \forall at \in \{st, wk\} : (s, o, write, +, at) \rightarrow (s, o, read, +, at)$.

Rule 3 The negation of the privilege to read an object implies the negation of the privilege to modify the object. Formally, $\forall s \in S, \forall o \in O, \forall at \in \{st, wk\} : (s, o, read, -, at) \rightarrow (s, o, write, -, at)$.

Rule 4 The create privilege on a database (class) implies the privilege to read the definition of the database (class). Formally, $\forall s \in S, \forall o \in Database \cup Class, \forall at \in \{st, wk\} : (s, o, create, +, at) \rightarrow (s, o, read_def, +, at)$.

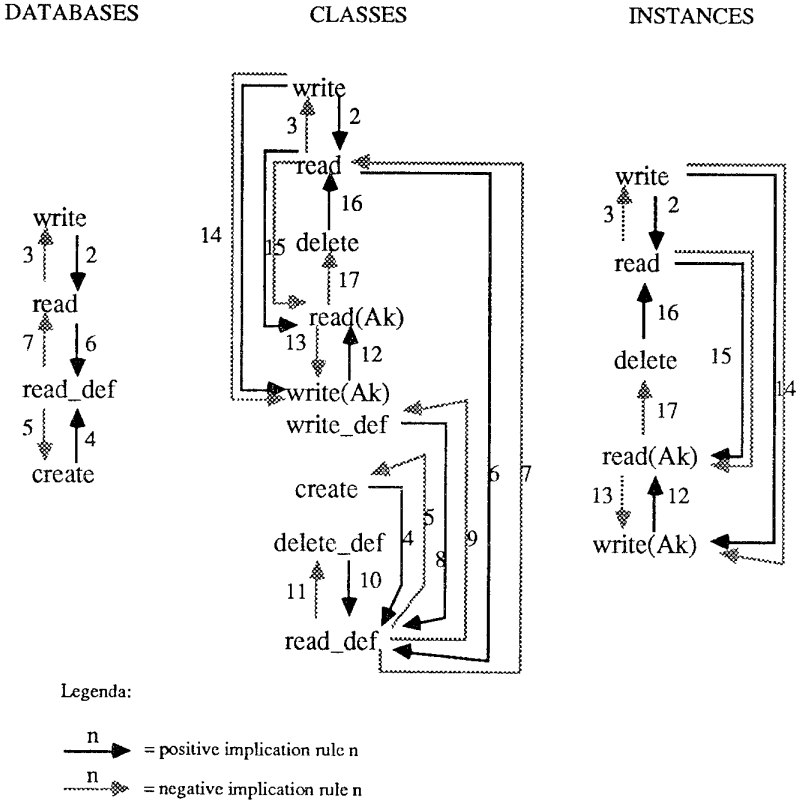


Figure 3: Graphical representation of the implication rules for access modes

Rule 4 states that if a subject is authorized to create a class within a database, then the subject is implicitly authorized to read the directory of the database. Likewise, if a subject is authorized to create an instance of a class, then the subject is implicitly authorized to read the definition of the class.

Rule 5 The negation of the privilege to read the definition of a database (class) implies the negation of the create privilege on the database (class). Formally, $\forall s \in S, \forall o \in Database \cup Class, \forall at \in \{st, wk\} : (s, o, read_def, -, at) \rightarrow (s, o, create, -, at)$.

Rule 5 states that if a subject is denied to read the directory of a database, then the subject is implicitly denied to create a class within the database. Likewise, if a subject is denied to read the definition of a class, then the subject is implicitly denied to create an instance of the class.

Rule 6 The read privilege on a database (class) implies the privilege to read the definition of the database (class). Formally, $\forall s \in S, \forall o \in Database \cup Class, \forall at \in \{st, wk\} : (s, o, read, +, at) \rightarrow (s, o, read_def, +, at)$.

According to Rule 6, the privileges to read all objects within a database and to read all instances of a class imply, respectively, the privileges to read the directory of the database and to read the definition of the class.

Rule 7 The negation of the privilege to read the definition of a database (class) implies the negation of the read privilege on the database (class). Formally, $\forall s \in S, \forall o \in Database \cup Class, \forall at \in \{st, wk\} : (s, o, read_def, -, at) \rightarrow (s, o, read, -, at)$.

According to Rule 7, the negation of the privileges to read the directory of a database and to read the definition of a class implies, respectively, the negation of the privileges to read all objects within the database and to read all instances of the class.

Rule 8 The privilege to modify the definition of a class implies the privilege to read the definition of the class. Formally, $\forall s \in S, \forall o \in Class, \forall at \in \{st, wk\} : (s, o, write_def, +, at) \rightarrow (s, o, read_def, +, at)$.

Rule 9 The negation of the privilege to read the definition of a class implies the negation of the privilege to modify the definition of the class. Formally, $\forall s \in S, \forall o \in Class, \forall at \in \{st, wk\} : (s, o, read_def, -, at) \rightarrow (s, o, write_def, -, at)$.

Rule 10 The privilege to delete the definition of a class implies the privilege to read the definition of the class. Formally, $\forall s \in S, \forall o \in Class, \forall at \in \{st, wk\} : (s, o, delete_def, +, at) \rightarrow (s, o, read_def, +, at)$.

Rule 11 The negation of the privilege to read the definition of a class implies the negation of the privilege to delete the definition of the class. Formally, $\forall s \in S, \forall o \in Class, \forall at \in \{st, wk\} : (s, o, read_def, -, at) \rightarrow (s, o, delete_def, -, at)$.

Rule 12 The privilege to modify an attribute on a class (instance) implies the privilege to read the attribute on the class (instance). Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall A_k \in Setof_attr(o), \forall at \in \{st, wk\} :$
$(s, o, write(A_k), +, at) \rightarrow (s, o, read(A_k), +, at)$.

Rule 12 states that the privileges to modify an attribute of an instance and to modify an attribute for all instances of a class imply, respectively, the privileges to read the attribute of the instance and to read the attribute for all instances of the class.

Rule 13 The negation of the privilege to read an attribute on a class (instance) implies the negation of the privilege to modify the attribute on the class (instance). Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall A_k \in Setof_attr(o), \forall at \in \{st, wk\} :$
$(s, o, read(A_k), -, at) \rightarrow (s, o, write(A_k), -, at)$.

Rule 13 states that the negation of the privileges to read an attribute of an instance and to read an attribute for all instances of a class implies, respectively, the negation of the privileges to modify the attribute of the instance and to modify the attribute for all instances of the class.

Rule 14 The write privilege on a class (instance) implies the write privilege for every attribute on the class (instance). The negation of the write privilege on an object propagates in the same way. Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall A_k \in Setof_attr(o), \forall as \in \{+,-\}, \forall at \in \{st, wk\} : (s, o, write, as, at) \rightarrow (s, o, write(A_k), as, at)$.

Rule 15 The read privilege on a class (instance) implies the read privilege for every attribute on the class (instance). The negation of the read privilege on an object propagates in the same way. Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall A_k \in Setof_attr(o), \forall as \in \{+,-\}, \forall at \in \{st, wk\} : (s, o, read, as, at) \rightarrow (s, o, read(A_k), as, at)$.

Rule 16 The delete privilege on a class (instance) implies the read privilege on the class (instance). Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall at \in \{st, wk\} :$
$(s, o, delete, +, at) \rightarrow (s, o, read, +, at)$.

Rule 17 The negation of the privilege to read an attribute on a class (instance) implies the negation of the delete privilege on the class (instance). Formally, $\forall s \in S, \forall o \in Class \cup Instance, \forall A_k \in Setof_attr(o), \forall at \in \{st, wk\} : (s, o, read(A_k), -, at) \rightarrow (s, o, delete, -, at)$.

4.3 Implication rules for objects

The implication rules for objects are based on the hierarchical structure of the set of objects (see, for example, Figure 2). Given an authorization for a privilege on an object o, these rules allow to derive an authorization for the same or a different privilege on objects contained in o.

In the following we enunciate the implication rules for objects. The graphical representation of these rules is given in Figure 4. The arcs of this graph represent the relationships among access modes applied to objects bound by the relationship of $<$.

Rule 18 The privilege to read a database (class) implies the privilege to read all the classes (instances) contained in it. The negation of the read privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database \cup Class, \forall o_j \in Class \cup Instance, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, read, as, at) \rightarrow (s, o_j, read, as, at)$.

Rule 19 The negation of the privilege to read the definition of a database implies the negation of the privilege to read the definition of every class of the database. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, read_def, -, at) \rightarrow (s, o_j, read_def, -, at)$.

Rule 20 The privilege to modify a database (class) implies the privilege to modify all the classes (instances) contained in it. The negation of the modify privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database \cup Class, \forall o_j \in Class \cup Instance, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, write, as, at) \rightarrow (s, o_j, write, as, at)$.

Rule 21 The write privilege on a database implies the delete privilege on each class of the database. The negation of the write privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i :$
$(s, o_i, write, as, at) \rightarrow (s, o_j, delete, as, at)$.

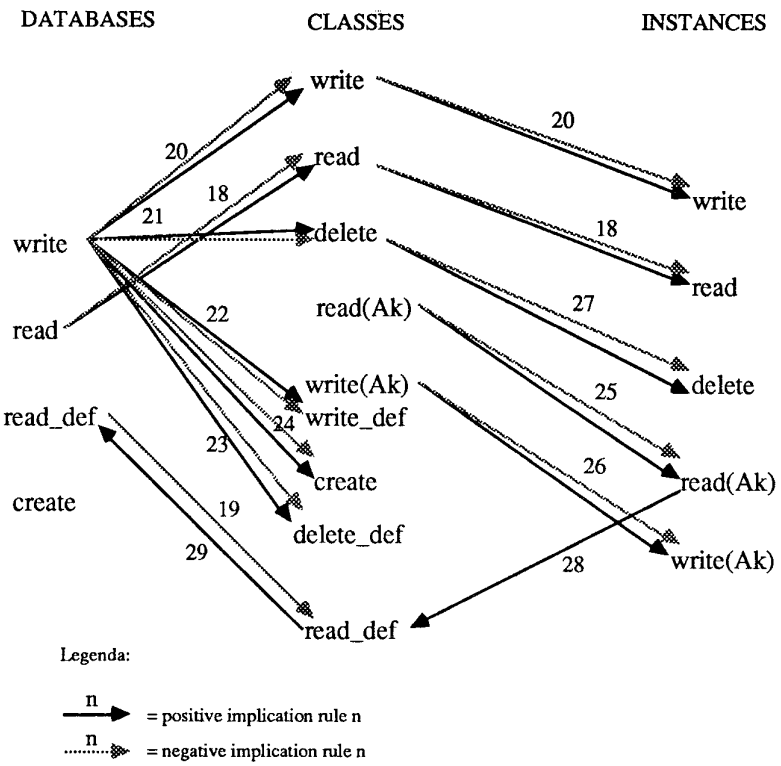


Figure 4: Graphical representation of the implication rules for objects

Rule 22 The write privilege on a database implies the privilege to modify the definition of each class of the database. The negation of the write privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, write, as, at) \rightarrow (s, o_j, write_def, as, at)$.

Rule 23 The write privilege on a database implies the privilege to delete the definition of every class of the database. The negation of the write privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, write, as, at) \rightarrow (s, o_j, delete_def, as, at)$.

Rule 24 The write privilege on a database implies the create privilege on every class of the database. The negation of the write privilege propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, write, as, at) \rightarrow (s, o_j, create, as, at)$.

Rule 25 The privilege to read an attribute on a class implies the privilege to read the attribute on every instance of the class. The negation of the privilege to read an attribute on a class propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Class, \forall A_k \in Setof_attr(o_i), \forall o_j \in Instance, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i,:$
$(s, o_i, read(A_k), as, at) \rightarrow (s, o_j, read(A_k), as, at)$.

Rule 26 The privilege to modify an attribute on a class implies the privilege to modify the attribute on every instance of the class. The negation of the privilege to modify an attribute on a class propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Class, \forall A_k \in Setof_attr(o_i), \forall o_j \in Instance, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i :$ $(s, o_i, write(A_k), as, at) \rightarrow (s, o_j, write(A_k), as, at)$.

Rule 27 The delete privilege on a class implies the privilege to delete every instance of the class. The negation of the delete privilege on a class propagates in the same way. Formally, $\forall s \in S, \forall o_i \in Class, \forall o_j \in Instance, \forall as \in \{+,-\}, \forall at \in \{st, wk\}, o_j < o_i : (s, o_i, delete, as, at) \rightarrow (s, o_j, delete, as, at)$.

All previous implication rules propagate authorizations top-down with respect to the object granularity hierarchy. We introduce now two implication rules that propagate authorizations bottom-up with respect to the object granularity hierarchy.

Rule 28 The privilege to read an attribute of an instance implies the privilege to read the definition of the class to which the instance belongs. Formally, $\forall s \in S, \forall o_i \in Class, \forall A_k \in Setof_attr(o_i), \forall o_j \in Instance, \forall at \in \{st, wk\}, o_j < o_i :$
$(s, o_j, read(A_k), +, at) \rightarrow (s, o_i, read_def, +, at)$.

Rule 29 The privilege to read the definition of a class implies the privilege to read the definition of the database which contains the class. Formally, $\forall s \in S, \forall o_i \in Database, \forall o_j \in Class, \forall at \in \{st, wk\}, o_j < o_i :$
$(s, o_j, read_def, +, at) \rightarrow (s, o_i, read_def, +, at)$.

4.4 Implication rules along the inheritance hierarchy

Implication rules along the inheritance hierarchy allow, given an authorization for a privilege on a class, to derive authorizations for the privilege on the subclasses of the class. Implication of authorizations along the inheritance hierarchy may be desirable in some cases and non desirable in other cases [13, 6, 10]. Hence, we allow the user defining a class to indicate whether he wants implication of authorizations along the inheritance hierarchy. If so, the authorizations of users on the superclasses for the create and delete access mode and for reading and writing attributes propagate to the class. The reason why privileges on the definition do not propagate is that these privileges should be reserved to the creator of the class. Note however that if a user receives a privilege on the class he indirectly receives the *read_def* privilege on the class, according to the implication rules of Section 4.2.

To determine whether a subclass inherits the authorizations from a superclass, we introduce function $Is_Inh(o', o)$ which, given a class o' and one of its superclasses o returns `True` if o' inherits the authorizations specified on o; returns `False`, otherwise.

Note that authorizations applicable to a specific attribute can be propagated to a subclass only if the attribute is inherited by the subclass. To determine this, we introduce a function $Attr_Inh(A_k, o', o)$ which, given an attribute A_k and classes o and o' returns `True` if o' inherits A_k from o; returns `False` otherwise.

Then, the implication of authorizations along the inheritance hierarchies is determined according to the following rules.

Rule 30 The authorization (negation) to create and delete on a class o propagates to all subclasses o' of o for which function $Is_Inh(o', o)$ returns **True**. Formally, $\forall s \in S, \forall o, o' \in Class, \forall m \in \{create,\ delete\}, \forall as \in \{+, -\}, \forall at \in \{st, wk\}, o' \prec o, Is_Inh(o', o) =$ **True**: $(s, o, m, as, at) \rightarrow (s, o', m, as, at)$.

Rule 31 The authorization (negation) to read/write an attribute A_k on a class o propagates to all the subclasses o' of o such that functions $Is_Inh(o', o)$ and $Attr_Inh(A_k, o', o)$ return **True**. Formally, $\forall s \in S, \forall o, o' \in Class, \forall m \in \{read, write\}, \forall A_k \in Setof_attr(o)$, $\forall as \in \{+, -\}, \forall at \in \{st, wk\}, o' \prec o, Is_Inh(o', o) =$ **True**, $Attr_Inh(A_k, o', o) =$ **True**: $(s, o, m(A_k), as, at) \rightarrow (s, o', m(A_k), as, at)$.

The implication rules along the inheritance hierarchy are illustrated in Figure 5. Label **E** associated with the arcs indicates the inheritance relationship.

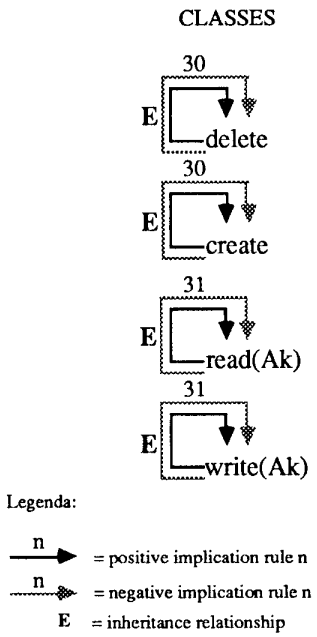


Figure 5: Graphical representation of the implication rules along the inheritance hierarchy

4.5 Derivation of implicit authorizations

Given the authorizations specified by the users (explicit), the rules illustrated in the previous section allow new authorizations (implicit) to be derived. The rules given for the different domains can be jointly applied thus allowing, from an authorization, the derivation of other authorizations, with same or different subject, same or different object, and same or different access mode.

Figure 6 illustrates the graph representing all the implication rules of our model, except for the rule specified for the set of subjects. In this graph, we use one-colored

arcs. An arc labeled with + (−) represents the implication for a positive (negative) authorization. For sake of clarity, the arcs are not labeled with the corresponding rule number.

Derivation of authorizations according to the rules corresponds to traversing the arcs of the graph. The set of authorizations implied by an explicit authorization a is called the extension of a. We now illustrate how to determine the extensions of strong and weak authorizations.

Since strong authorizations do not admit exceptions, the extension of a strong authorization is composed of all authorizations which can be derived from a by applying the rules. This is formalized by the following definition.

Definition 6 (Extension of a strong authorization) The *extension* of a strong authorization a is the set $E(a)$ defined as: $E(a) = \{a' \mid a \rightarrow_m a', m \geq 0\}$.

Note that a belongs to this set ($m = 0$).

Determining the extension of a weak authorization is more complex. Indeed, since weak authorizations can have exceptions, the set of authorizations derivable from an explicit weak authorization depends also on the content of the strong and weak authorization bases. Hence, before defining the extension of a weak authorization we give some definitions to determine when a weak authorization is overridden by exceptions.

Definition 7 (More specific subject) Given two subjects $s, s' \in S$, s' is *more specific* than s, written $s' \lhd s$, if and only if $s' <_n s$ with $n \geq 1$.

The above definition states that subject s' is *more specific* than subject s if and only if s' is a member (direct or indirect) of s. In the following, notation $s' \trianglelefteq s$ indicates that either $s' = s$ or $s' \lhd s$.

Definition 8 (More specific object) Given two objects $o, o' \in O$, o' is *more specific* than o, written $o' \lhd o$, if and only if either $o' <_n o$ or $o' \prec_n o$, with $n \geq 1$.

Definition 8 states that object o' is *more specific* than object o if either o' is descendant of o in the object granularity hierarchy, or o' is a subclass of o.

Definition 9 (More specific access mode) Given two access modes $m, m' \in M(class) \cup M(instance)$, m' is *more specific* than m, written $m' \lhd m$, if and only if $m' \in \{write(A_k), read(A_k)\}$, and $m \in \{write, read\}$.

Definition 9 states that an access mode referred to a single attribute is more specific than an access mode referred to a set of attributes. Note that according to Definition 9 the write access mode on an attribute is considered more specific than the read access mode on an instance. Moreover the read access mode on an attribute is considered more specific than the write access mode on an instance. These relationships hold because of the implication relationship existing between the two access modes (Section 4.2).

Definition 10 (More specific authorization) Given two authorizations $a, a' \in$ WAB, authorization a' is *more specific* than authorization a, written $a' \lhd a$, if and only if any of the following conditions is satisfied:

1. $s(a') \trianglelefteq s(a), o(a') = o(a), m(a') \lhd m(a)$;

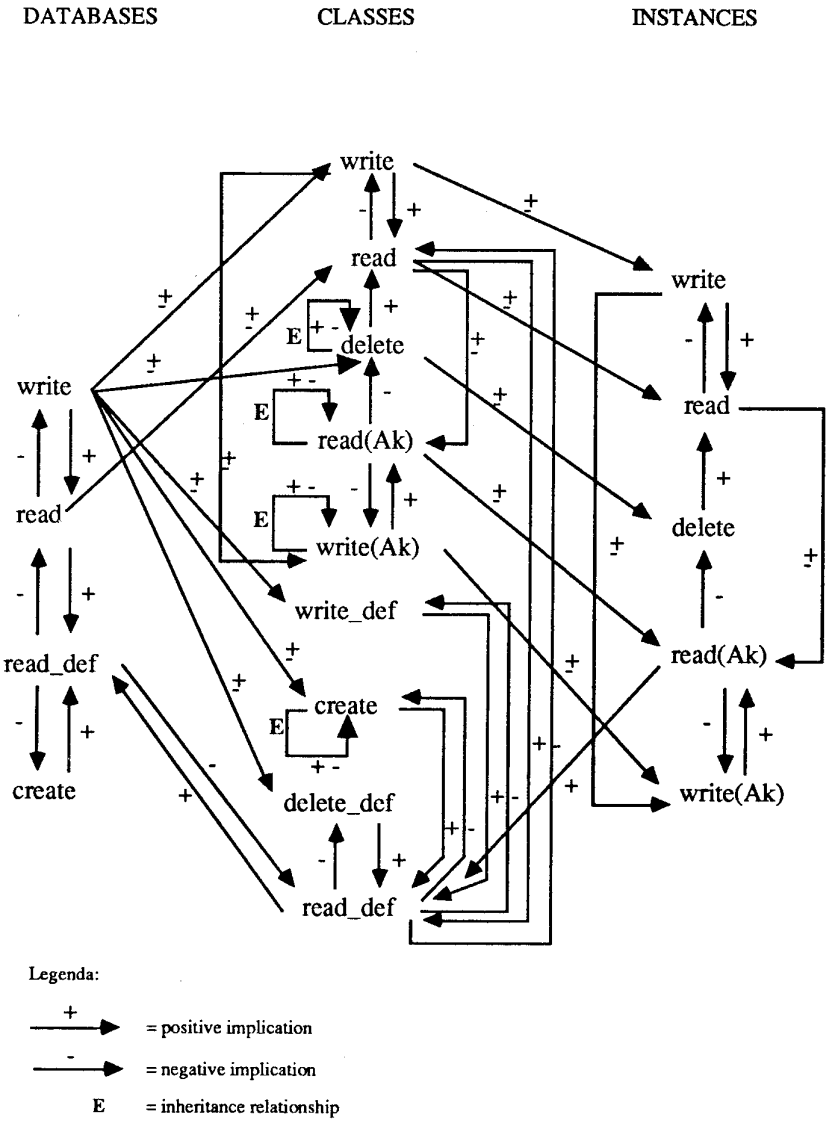


Figure 6: Graphical representation of the implication rules of our model

2. $s(a') \trianglelefteq s(a), o(a') \lhd o(a)$;
3. $s(a') \lhd s(a), o(a') = o(a), m(a) \not\lhd m(a')$.

Definition 10 states that an authorization a' is *more specific* than an authorization a if and only if either (1) the subject of a' is more specific than or equal to the subject of a, the object of a and a' are equal, and the access mode of a' is more specific than the access mode of a; (2) the subject of a' is more specific than or equal to the subject of a, and the object of a' is more specific than the object of a; or (3) the subject of a' is more specific than the subject of a, the objects of a and a' are equal, and the access mode of a is not more specific than the access mode of a'. The last condition of item (3) is needed to avoid, given two authorizations a and a' such that $s(a') \lhd s(a)$ and $o(a') = o(a)$, considering $a' \lhd a$ if $m(a) \lhd m(a')$. For example, given authorizations a = (G_2,Empl,*read*(Name),+, wk) and a' = (Bob,Empl,*read*,−, wk), a' cannot be considered more specific than a since $s(a') \lhd s(a)$ (Bob $\lhd G_2$), $o(a') = o(a)$ (= Emp_1), but $m(a) \lhd m(a')$ (*read*(Name) $\lhd$ *read*).

Example 1 Consider authorizations:

- a_1 = (G_2,Employees,*write*,+, wk);
- a_2 = (Bob,Employees,*read*(Salary),−, wk);
- a_3 = (G_2,Emp_1,*write*,+, wk);
- a_4 = (Bob,Employees,*delete*,−, wk).

$a_2 \lhd a_1$ since $s(a_2) \lhd s(a_1), o(a_2) = o(a_1), m(a_2) \lhd m(a_1)$ (Definition 10, item 1).
$a_3 \lhd a_1$ since $s(a_3) = s(a_1), o(a_3) \lhd o(a_1)$ (Definition 10, item 2).
$a_4 \lhd a_1$ since $s(a_4) \lhd s(a_1), o(a_4) = o(a_1), m(a_1) \not\lhd m(a_4)$ (Definition 10, item 3).

We now define when a weak authorization is overridden. We distinguish the cases where a weak authorization is overridden by a strong authorization or by another weak authorization.

In the following, given an authorization $a = (s, o, m, as, at)$, notation $|a|$ denotes the set composed of authorization a and its negation (i.e., an authorization with the same subject, object, access mode, and type as authorization a, but with different sign). That is, $|a| = \{(s, o, m, +, at), (s, o, m, -, at)\}$. Given a weak authorization $a = (s, o, m, as, wk)$, $|a|^f$ denotes the set composed of authorization authorization a' and the negation of a', where a' is a *strong* authorization with the same subject, object, access mode, and sign as a. That is, $|a|^f = \{(s, o, m, +, st), (s, o, m, -, st)\}$.

Definition 11 (Strong overriding) Given two authorizations a, a_k such that $a \in ASP, at(a) = wk, a_k \in$ SAB, we say that a_k *overrides* a, written $a_k \gg a$ if and only if $\exists a_i \in |a|^f, a_k \rightarrow_m a_i, m \geq 0$.

Definition 11 states that a strong authorization a_k overrides a weak authorization a if a_k implies (or is equal to, if m=0) an authorization with same subject, object and access mode as a and different type.

The overriding relationship among weak authorizations is more complex. In particular, in order to define whether an authorization a_k overrides an authorization a, the authorization from which a has been derived must be considered. The overriding relationship among weak authorizations is formalized by the following definition.

Definition 12 (Weak overriding) Given three authorizations a, a_k, a' such that $a, a_k \in$ WAB, $a \rightarrow_\ell a', \ell \geq 1, s(a_k) = s(a'), o(a_k) = o(a')$, a_k *overrides* a', written $a_k \gg a'$ if and only if $\exists a_i \in |a'|, a_k \rightarrow_n a_i, n \geq 0, a_k \lhd a$.

Definition 12 states that an explicit weak authorization a_k overrides an authorization a' implied by an explicit weak authorization a if and only if the subject and the object of a_k are equal to the subject and the object of a', authorization a' or its negation is implied authorization a_k, and the authorization a_k is more specific than authorization a.

Given the above definitions, the extension of a weak authorization is defined as follows.

Definition 13 (Extension of a weak authorization) The *extension* of a weak authorization a is the set $E(a)$ defined as: $E(a) = \bigcup_{n=0}^{n=N_a} E_n(a)$

where

$$E_0(a) = \begin{cases} \{a\} & \text{if } \nexists a_k \in \text{SAB}, a_k \gg a \\ \emptyset & \text{otherwise} \end{cases}$$

$E_n(a) = \{a' \mid \exists a'' \in E_{n-1}(a), \nexists a_k \in (\text{SAB} \cup \text{WAB}), a'' \rightarrow a', a_k \gg a'\}$,
and
N_a is the first n such that $E_{n+1}(a) = \emptyset$.

The existence of such an N_a is ensured by the fact that the implication rules are finite and they work on finite lattices.

Definition 13 states that the extension of a weak authorization a is composed of all the authorizations which can be derived by a and which are not overridden. Note that if $a \notin E_o(a)$, then set $E(a)$ is empty.

Example 2 Consider the subject graph illustrated in Figure 1 and the object hierarchy illustrated in Figure 2. Suppose that SAB = $\emptyset$, WAB = $\{a_1, a_2\}$ where:

- a_1 = (G_6,Employees,*read*,+, *wk*);
- a_2 = (Mary,Empl,*read*,−, *wk*).

Let us analyze how authorization a_1 propagates. For sake of clarity we do not derive all extension of a_1 but only some authorizations. First of all, $a_1 \in E(a_1)$ since no strong authorization exists which overrides a_1. Moreover, $a_3, a_4 \in E(a_1)$ where:

- a_3 = (G_6,Empl,*read*,+, *wk*);
- a_4 = (Mary,Employees,*read*,+, *wk*).

Indeed, $a_1 \rightarrow a_3, a_1 \rightarrow a_4, a_1 \in E(a_1)$ and no authorization exists which overrides a_3 or a_4. By contrast, authorization a = (Mary,Empl,*read*,+, *wk*) $\notin E(a_1)$. Even if $a_3 \rightarrow a, a_3 \in E_1(a_1)$ and $a_4 \rightarrow a, a_4 \in E_1(a_1)$, $a \notin E(a_1)$ since $a_2 \in$ WAB and $a_2 \gg a$. In fact, according to Definition 12, $a_2 \rightarrow_0 a'$ = (Mary,Empl,*read*,−, *wk*), $a' \in |a|$, and $a_2 \lhd a_1$. This last relationship comes from Definition 10 since $s(a_2) \lhd s(a_1)$ (Mary $\lhd G_6$) and $o(a_2) \lhd o(a_1)$ (Emp$_1$ $\lhd$ Employees).

Figure 7 illustrates the propagation of authorization a_1.

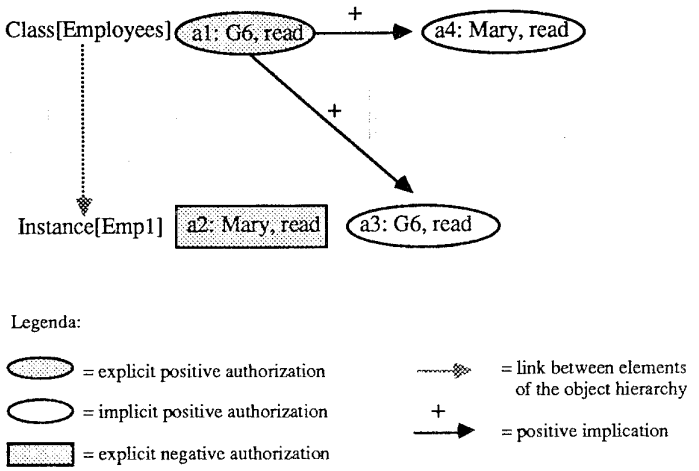


Figure 7: An example of derivation of implicit authorizations

5 Authorization State

The authorizations valid at a given time are all the authorizations explicitly defined by the users, or derived from the authorizations explicitly defined by the users, which are not overridden. The set of authorizations valid at a given time is called the *authorization state*, formally defined as follows.

Definition 14 (Authorization State) The *authorization state* $AS \subset ASP$ is a set of authorizations defined as follows:

$$AS = \bigcup_{a \in \text{SAB} \cup \text{WAB}} E(a)$$

The simultaneous presence of two (explicit or implicit) authorizations a and a' equal but for the sign ($a' = \neg a$) and such that no one overrides the other is interpreted as an inconsistency in our model. This is formalized by the following definition.

Definition 15 (Consistent authorization state) An authorization state AS is *consistent* if and only if $\nexists a, a' \in AS$ such that $a' = \neg a$.

Inconsistencies are not allowed in our model and, accordingly, the following invariant must hold.

Property 1 (Consistency of the AS) *The authorization state is consistent.*

Each time an authorization is granted, the system determines whether the insertion of the authorizations would introduce an inconsistency in the authorization state. If so, the grant operation is rejected.

Example 3 Consider to the subject graph illustrated in Figure 1 and the object hierarchy illustrated in Figure 2. Suppose that SAB = $\emptyset$, WAB = $\{a_1, a_2\}$ where:

- a_1 = (Bob,Administration,*read_def*,−, *wk*);
- a_2 = (Bob,Emp_2,*read*(Address),+, *wk*).

Authorization a_1 states that user Bob is denied to read the directory of the database Administration; authorization a_2 states that user Bob is authorized to read the attribute Address of instance Emp_2 of class Employees.

Those authorizations generate an inconsistency in the authorization state, since authorizations $a_3 \in E(a_1)$ and $a_4 \in E(a_2)$ where:

- a_3 = (Bob,Employees,*read_def*,−, *wk*),
- a_4 = (Bob,Employees,*read_def*,+, *wk*),

which are one the negation of the other, belong to the authorization state (Figure 8).

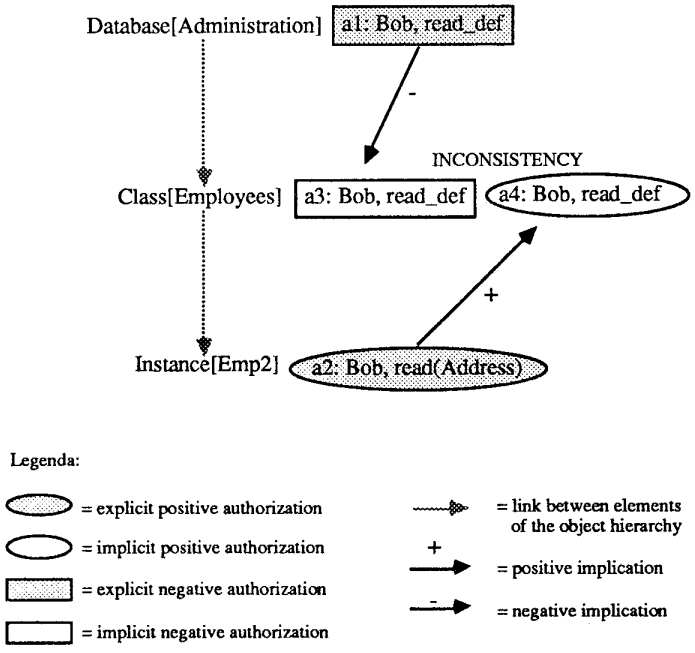


Figure 8: An example of inconsistency in the authorization state

6 Access control

In this section, we illustrate how access control is performed. An access request can be characterized as a 3-tuple $\langle u, o, m \rangle$ with $u \in U, o \in O, m \in M$, indicating that user u requests to exercise access mode m on object o.

The access control determines whether fully grant, partially grant, or deny the access to the user. Consider a request $\langle u, o, m \rangle$. The access control performs the

following steps. First, the system checks whether m is executable on o (for instance, the *read_def* access mode on a class allows to read the definition of the class) or on its components (for instance, the *read* access mode on a class allows to read all attributes of all instances of the class). In the first case, the access request is granted if there exists in the authorization state a positive authorization which satisfies the request, and it is rejected otherwise. In the second case, the access request specified by the user is split into a set of elementary access requests with u as subject, a component of o as object, and an access mode related to m (according to the rules) as access mode. If all elementary access requests are authorized (i.e., a positive authorization exists for each of them in the authorization state), the system fully grants the access to the user; if none of the elementary access requests are authorized, the system denies the access to the user; if only some of the elementary access requests are authorized, the system partially grants the access to the user returning the subset of elementary access requests that are authorized.

The access control can be represented by a function b defined as follows:

$$b : U \times O \times M \rightarrow 2^{U \times O \times M} \times \{\texttt{True},\texttt{False}\}.$$

Given the access request $\langle u, o, m\rangle$, function b returns $\langle (u, o, m), \texttt{True}\rangle$ if the access request is fully granted; it returns $\langle (u, o, m), \texttt{False}\rangle$ if the access request is rejected; it returns the set $\{\langle (u, o_i, m_k), \texttt{True}\rangle\}$ (i.e., the set of elementary access requests authorized) if the access request specified by is partially granted.

Example 4 Consider the authorization state illustrated in Figure 7 and access request ⟨Mary,Employees,*read*⟩ which states that user Mary requests to read all instances of class Employees.

The function of access control b returns the following set: $\{\langle$(Mary,Emp$_i$,*read*),True$\rangle\}$ for all $i \neq 1$. In fact, user Mary is implicitly authorized to read all instances of class Employees, except for instance Emp1. Thus, the access request ⟨Mary,Employees,*read*⟩ is partially granted.

7 Conclusions

The semantic concepts of object-oriented database systems make traditional access control model developed for operating systems and traditional databases systems inadequate. Work in the area of authorization models for object-oriented systems is still at a preliminary stage and many questions are left open. In this paper we have presented an authorization model for object-oriented databases. The model supports both positive authorizations (meaning permission to do something) and negative authorizations (meaning denial to do something). From the authorizations specified by the users the system derives new authorizations on the basis of the relationships existing among subjects, objects, access modes, and on the inheritance hierarchies. The model allows authorizations to be overridden by permitting two types of authorizations (strong and weak). Strong authorizations cannot be overridden whereas weak authorizations can be overridden according to specified rules. In the paper we have given the rules for the derivation and the overriding of authorizations.

References

[1] R. AHAD ET AL., "Supporting access control in an object-oriented database language," *Proc. Third International Conference on Extending Database Technology (EDBT)*, Vienna (Austria), Springer-Verlag Lecture Notes in Computer Science, Vol. 580, 1992, pp. 184–200.

[2] E. BERTINO, C. BETTINI, AND P. SAMARATI, "A temporal authorization model," submitted for publication.

[3] E. BERTINO, F. ORIGGI, AND P. SAMARATI, "An extended authorization model for object-oriented databases," in preparation.

[4] E. BERTINO AND L. MARTINO, *Object-Oriented database systems: concepts and architectures*," Addison-Wesley International, 1993.

[5] E. BERTINO, S. JAJODIA, P. SAMARATI, "Access Controls in Object-Oriented Database Systems: Some Approaches and Issues," *Advanced Database Concepts and Research Issues*, N.Adam and B. Bhargava, eds., LNCS 759, Springer- Verlag, 1993.

[6] E. BERTINO AND H. WEIGAND, "An approach to authorization modeling in object-oriented database systems," *Data and Knowledge Engineering*, Vol. 12, No. 1, February 1994, pp. 1–29.

[7] R. BREITL, ET AL., "The GemStone data management system," in *Object-Oriented Concepts, Databases, and Applications*, W. Kim, and F. Lochovsky, eds., Addison- Wesley, 1989, pp. 283–308.

[8] H.H. BRÜGGEMANN, "Rights in an object-oriented environment," in *Database Security, V: Status and Prospects*, C.E. Landwehr and S. Jajodia, eds., North-Holland, Amsterdam, 1992, pp. 99–115.

[9] K. DITTRICH, M. HARTIG, AND H. PFEFFERLE, "Discretionary access control in structurally object-oriented database systems," in *Database Security, II: Status and Prospects*, C.E. Landwehr, ed., North-Holland, Amsterdam, 1989, pages 105–121.

[10] E. B. FERNANDEZ, E. GUDES, H. SONG, "A model of evaluation and administration of security in object-oriented databases," in *IEEE-TKDE*, vol. 6, no. 2, April 1994.

[11] L.J. GALLAGHER, "Object SQL: language extensions for object data management," in *Proc. First International Conference of Information and Knowledge Management (CIKM)* , Baltimore, Maryland, November, 1992.

[12] W. KIM, E. BERTINO, J.F. GARZA, "Composite object revisited," in *Proc. ACM-SIGMOD International Conference on the Management of Data*, Portland (Oreg.), May-June 1989.

[13] F. RABITTI, E. BERTINO, W. KIM, AND D. WOELK, "A model of authorization for next-generation database systems," *ACM Trans. on Database Systems*, Vol. 16, No. 1, March 1991, pp. 88–131.

The Integration of Security and Integrity Constraints in MOKUM

Reind P. van de Riet
Jack Beukering

Department of Mathematics and Computer Science
Vrije Universiteit
Amsterdam

e-mail: {vdriet, jbeuker}@cs.vu.nl
March, 1994

Abstract

In this paper we will describe how constraints involving integrity and security can be specified in the active object-oriented knowledge-base system MOKUM. Also we will indicate how they are implemented.
Keywords: Security and Database systems, integrity constraints, Object-Oriented Databases, Active knowledge bases.

Contents

1 Introduction **3**

2 About MOKUM: two levels **4**
2.1 MOKUM in a nutshell 4
2.2 About objects, collections, types an classes 5
2.3 Some details about the implementation 6

3 About constraints **6**
3.1 classification of constraints 7
3.2 Security constraints 8

4 Constraints in MOKUM scripts **9**
4.1 Scripts in MOKUM 9
4.2 Security in MOKUM 12
4.3 Visibility and accessability in MOKUM 13

5 Constraints as MOKUM restrictions **16**

6 Conclusions **18**

1 Introduction

In a knowledge base system constraints are very important entities which should be treated carefully. In particular for active knowledge base systems which are supposed to realize Information- and Communication Systems (ICS), which connect people and information systems. One has to accommodate for the fact that people willfully and unwillfully are changing data and rules about the data. It is of paramount importance that these changes are governed by rules in the form of constraints and security checks, which are maintained by the knowledge base system in a safe way. For the quality of the data the constraints are important, while access rules must give security in order to be able to guarantee protection and privacy of people's data. Both kinds of rules are interrelated very much, notified already in one of the earliest papers by the Ingres group: [Stonebraker,Wong & Held, 1976] and which is not been taken care of by most data- and knowledge base systems: constraints on the data are globally specified in the form of triggers, while security is defined using a single access matrix. See e.g. the overview paper [Grefen & Apers, 93] which deals with integrity constraints in database systems and the paper [Paton,Diaz & Barja, 93] about constraints in the form of rules in Object-Oriented (O-O) systems.

Our current work on integrity and security in databases can be considered as a continuation of work by our group in the not so recent past on security and databases. We explicitly refer here to the work on:

* privacy and security and a programming language approach; see [vandeRiet, Kersten & Wasserman, 82], [vandeRiet, Kersten & Wasserman, 82], [Wasserman, vandeRiet, Kersten & Leveson, 1983]

* access control; see [Kersten,85],

* keeping secrets by a knowledge base [Sicherman, deJonge & vandeRiet, 83],

* statistical databases [deJonge, 83] and

* cryptography [deJonge, 85].

In the more recent years our attention was focused on Object-Oriented systems. We developed the MOKUM [1] system, which is an active object-oriented knowledge base system (see [vandeRiet,89]). Also more formal aspects of Conceptual Modelling we worked on. The current paper can be seen as a study of the issue of Security within an existing system as seen from the standpoint of a Conceptual Model. We will describe how constraints involving integrity and security can be specified in MOKUM. In MOKUM both kinds of constraints are treated from one viewpoint and are not separated. The basic mechanism for communication in MOKUM is the message, sent from object to object. It can always be seen who the sender of a message is. Objects who send and receive messages can usually be identified with office employees having certain rights and responsibilities. In this way MOKUM differs from usual O-O systems where objects denote active pieces of software. In most other O-O systems constraints on classes and meta-classes are considered themselves as active objects who become active.This is not the case in MOKUM. We have deliberately designed MOKUM to be close to the real world applications, where persons and institutions can be active, but where regulations, specifications or collections of things cannot be active.

[1] The acronym MOKUM stands for Manipulating Objects with Knowledge and Understanding in Mokum (=Amsterdam).

In section 2 we will give a short introduction of the MOKUM architecture. In section 3 we will give some taxonomy of constraints and security rules. In sections 4 and 5 we will see how these rules can be treated in MOKUM, first in the form of scripts, then in the form of restrictions. In section 4 we will also focus our attention to security constraints, how they can be implemented in MOKUM and some theory about them. In the final section we will give some conclusions and describe the status of the MOKUM system.

```
type  person is_a  thing
        has_a      name: string
    script  part of person endscript
type  employee is_a  person
        has_a      mgr: employee
      private
        has_a      salary: int
    script  part of employee endscript
type  empl_adm is_a  employee /*shorthand for employee_administrator*/
      private
        has_a      nr_of_employees:int
        has_a      employees: collection_of  employee
    script  part of empl_adm endscript
```

Figure 1: Three type definitions in MOKUM.

2 About MOKUM: two levels

2.1 MOKUM in a nutshell

In Information Systems it is customary to differentiate between intension and extension. The intension is the form of the IS, called schema or Conceptual Model, while the extension is the contents of the IS, i.e. the collection of all objects in the IS.

In MOKUM we have made a principle of this division: we have type definitions and instances of these types in the form of objects. Each object is the instance of one (or more) type(s). As shown in fig. 1, objects of type employee are also objects of type person.We adopt here the convention that all identifiers which have a pre-defined meaning in MOKUM are printed in bold face. For an example of a script see figure 2 in a next section.

Objects are made by **new** . After their creation they have type thing, after which their only property is that they have an identity: the object_identity, which is a uniquely determined identifier, in principle not accessible in the outside world. After being created they can get other types, such as person or employee. For example, a new person, later also becoming an employee, identified by the value of the (Prolog) variable John is created after:

```
new (John), add type (John, person, [(name, 'Jan')]),
add_type (John, employee, [(salary, 10.000)])
```

As customary, we say that objects having a certain type are instances of that type. Because objects have types they have attributes, which may have values.

In our case, John is an instance of person as well as of employee so that it has the attributes name, salary and mgr, of which only the first two are filled in. One can give a value to an attribute as follows:

Pete **to** John:mgr

Asking values of attributes can be done as follows:

S **from** John:salary

Because salary is specified as **private** this statement is protected: only in the script part of the employee type of John (and in the script parts of its possible subtypes), this statement will result in putting John's salary in the Prolog variable S. Outside this script part (with the exception to be discussed in connection with keepers) this statement will fail. Also this statement will be successful only when the object who asks for the salary is John itself (or its keeper, see next discussion). For a more detailed discussion about access protection see section 4.3.

In a type definition one also can specify restrictions, computed attributes and scripts. For restrictions and scripts we refer to sections 4 and 5; for computed attributes we refer to other MOKUM documents (See e.g. [vandeRiet, 89]).

2.2 About objects, collections, types an classes

With types one can reason (epistemic). So inheritance of properties is defined on the level of types: from the above definitions one can infer that objects of type employee also have the person attribute name. As far as the object is concerned, there is no difference between John having salary as attribute and name. In our perspective it would be wrong to say that the object John in any way inherits some properties.

Objects stand for entities existing in the Universe of Discourse, which may be put together in collections (ontology), which also have a long life as they are persistent and as such stored in an O-O database. In most O-O systems the collection of all objects being instances of a certain type is called a class, usually with the same name. This is confusing, in particular when that class is also called an object. A consequence of this principle is that MOKUM does not have the notion of class, but it has the notion of collection.

Definition: collection and its keeper
A Collection is a set of objects having a certain type, and it can be defined only as the value of an attribute of an object called its keeper.

It is the task of the keeper of a collection C to maintain security and integrity rules about C and its members.

The elements of C all have at least one common type. (This condition is not a severe one, because, if wants to put objects of different types in C, another type can be introduced, being a supertype of these different types (and one could take thing for it), and these objects can be made instances of that supertype.

Other attributes of K can be connected to C, e.g. an attribute nr_of_elmts can

denote the current number of elements in C, while an attribute max_nr_of_elmts can denote the maximum number of elements in C. The combination collection and special attributes is what other people call "grouping". (see [Motschnig & Storey, 93])

In the example above we may have several employee administrators, all instances of the type empl_adm. All have a (different) collection of employee objects for which they are responsible. These collections are also protected. In the script part and in restrictions the access to these collections and their integrity constraints is regulated.

In the script part the reaction of an object is specified upon receiving some message. The object can be in different states and dependent on the state it reacts on messages called triggers. The behaviour of an object can be characterized as a Finite State Automaton. There are two kinds of triggers, one is activated by another object, called sender, and the other is a reaction on a timer, set by the object itself. The message itself is also an object and identified by *message*; it must be an instance of a user-defined type, being itself a subtype of the type message_type. To transfer parameters in the message one can simply use attributes. In section 4 we will see some examples. For more extensive examples one is referred to [vandeRiet, 89].

2.3 Some details about the implementation

The current MOKUM system consists of a compiler, a kernel, a storage facility and an animation facility. The compiler translates a knowledge base system written in the MOKUM syntax into Prolog predicates. These predicates together with the kernel form a Prolog program which can be run as a simulation for a real ICS system. Objects can be stored in an INGRES database system, (see [vandeRiet & Gamito, 90]) but not necessarily, they can also be stored in the Prolog fact base. The animation facility (see [Croshere,vandeRiet & Blom, 93]) makes it possible to see what is happening during a MOKUM simulation. Objects are shown, for which three windows are used, one is the animation window in which one can see the objects changing. The system is really very small (about 20 pages of Prolog text for the kernel, 20 pages of C code for the interface with INGRES, 20 pages of XPC Prolog text for the animation facility and also about 20 pages for the compiler, written in C). It is mainly been used in an educational environment, where it should be easy for students to add facilities, such as the facility discussed in this paper.

The addition of the restrictions and constraints resulted in a system called MOKUM-C.

3 About constraints

A constraint is a formula which specifies some properties of some collections of objects. In its most general form several collections may be involved, while the constraint may refer to the set properties, such as nr_of_elmts, or combinations of sets, such as the union or the intersection of two sets, but also to properties of the individual elements, such as the salary in case of a collection of millionaires. Furthermore, a constraint may refer to a certain operation, e.g. an update, or the addition of an object to a collection and the constraint may refer to both the value before and the value after the operation. Finally, a constraint may refer to the context in which it should be seen, such as a the object who issues

the update.

In general, one can say that each time some manipulation on an object or a collection mentioned in a constraint or an object being an element of such a collection is performed (create/change/inspect/destroy), some check has to be performed.

3.1 classification of constraints

In the following we will give a classification of the different kinds of constraints possible.

First, there are constraints which refer only to static properties of the objects, i.e. properties which must always hold, as soon as an object or collection gets a certain type (i.e. has indeed properties about which the constraint deals).

Second, there are dynamic properties which are attached to certain operations; these properties deal with the properties before and after the operation, for example that a salary may only increase.

Third, there is context dependency, usually the object on whose behalf the operation is carried out is attached to the operation. (In MOKUM terminology: the sender of the message). There may be constraints restricting these operations. Usually called authorization constraints. We are working in an environment that objects may stand for persons who want to change or inspect certain properties, such as their salary. We assume that real persons can sit at a workstation and be connected to their person object counterpart. One can imagine that the properties this real person can see and change are the person properties, but the properties as employee, such as salary, are properties not freely available. These properties are only available through the interference of the employee administrator.

The following is a possible list of different kinds of constraints:

C1 Constraints on attributes of one object only:

- **C1.1** single attribute constraint; example:
  $0 < age < 140$
- **C1.2** two-attribute constraint; example:
  $nr_of_elmts < max_nr_of_elmts$
- **C1.3** constraint concerning new and old value; example:
  $new.age > old.age$

C2 Constraints on attributes of two objects

C3 Constraints on one collection

- **C3.1** The properties of the members of the collection are not involved; example:
  $\sharp(C) < 10$
- **C3.2** Only the properties of the members of the collection are involved; example:

  $\forall_{x \in C} : x.salary > 1000.000$ or
  $\exists_{e \in C} : e.function = boss$
- **C3.3** A combination: example:

  $\sharp(C) < 10 \wedge \exists_{x \in C} : x.salary > 1000.000$

C4 Constraints on two or more collections C and D:

C4.1 intersection constraint: example:
$(C \cap D) = \emptyset$

C4.2 constraint involving a cardinality: example:
$\sharp(C \cup D) < 10$

The operations we have to look at are the following: Suppose an object O is also a member of a collection C.

OP1 only on the single object

OP1.1 O's creation

OP1.2 O gets a new type

OP1.3 some attribute of O gets a value, without having one before;

OP1.4 some attribute of O gets another value;

OP1.5 O is used as value of an attribute of (another) object

OP1.6 O is destroyed

OP2 only on the collection

OP2.1 C is created

OP2.2 C is destroyed

OP3 on the relation of the object and the collection

OP3.1 O is added into C

OP3.2 O is deleted from C

The constraints which involve old and new values are all attached to certain operations, usually update operations.

The above operations may all be connected to the actor of the operation, i.e. the object who issued the request to carry out this operation which must be entitled to do so. In principle the context can involve the time of the day and or conditions of other objects. Example: a nurse can inspect a file of a patient when the responsible physician is not present.

There is a large number of literature about the efficient maintenance of integrity and security constraints, we only refer to [Weigand, 93] for an overview.

3.2 Security constraints

When a constraint involves explicitly the context in which an operation is taken place we say that it is a security constraint. For static constraints there is no security involved, only when the constraints are dynamic one can speak of security. In this case some operation has to be carried out on some piece of the knowledge base and the operation is carried out on behalf of some entity, usually a person. The context is represented by the invoker of the operation and possibly some other circumstantial information, such as the time of the day, or mode of operation (e.g. urgency). A typical situation is the Automatic Teller Machine: dependent on the person who uses the ATM and dependent on the amount of cash available, money can be withdrawn, which leads to an update of the client's account in the bank's database. The operation is an update on the account of the database, the context consists of invoker and the state of the ATM.

In some security systems one has levels of security. For example, in a military environment the general can see and do more than a soldier. The information is characterized as top-secret, secret, confidential and non-confidential, say. We shall see in the next section how such a security system can be specified in MOKUM.

4 Constraints in MOKUM scripts

4.1 Scripts in MOKUM

In MOKUM there is a natural place where these checks can be executed, namely at the script part of the object or of the keeper of the collection involved in the constraints.

Let us assume that we have an employee object, John, member of a collection employees with keeper Empl_adm. John also has type person. See figure 1 for the type definitions.

We assume that salary is a private attribute which can only be seen and changed by the object itself and by the keepers of the collections the object is a member of. We assume that John as employee is a member of the employees attribute of the object Empl_adm. There are two places where it is allowed to manipulate the salary attribute of John: in its own script, i.e. the script part of employee and in Empl_adm's script, i.e. the script part of empl_adm.

In principle, the compiler can syntactically check that indeed at no other place "salary" occurs. This is not enough however as the compiler cannot see that John is referring his own salary. So a dynamic check is also necessary, and this is how the kernel of MOKUM works: operations on private attributes are translated so that this check is carried out. To be more precise: the operation **to** , which puts a new value in some attribute of an object O, also has as parameter the object who is the caller, and when the attribute is a private one it is checked whether the caller object is the same as the object O. Or, whether the caller object is among the keepers of O. Evidently, this presupposes that we have available for every object a list of its keepers. Things are somewhat more complicated because also the type of the collection has to be administered. It is possible that a keeper has several types and it is also possible that it is the keeper of several collections, even of several types. In actual practice when objects are stored in a separate database the whole of operation and checking is left to the database system, using some kind of access table.

In any way, on a low level, MOKUM checks whether a certain operation can be performed by some object in its script part. This provides the means for a full and most general checking for the context in which some operation referring to a private attribute is carried out. A message must be sent to the keeper of the object, or to the object itself. This message should specify the kind of operation to be performed.

Let us assume for the above example, that there is no script part for employee, i.e. the employee object can only be operated upon by Empl_adm. We may have as a rule that an object of type person can only see its own salary while its manager can change (and for the sake of the example: increase) it. In figure 2 the script of empl_adm is shown which specifies these rules:

Note that the commands in figure 2 should be read as Prolog predicates: if one fails the rest is not executed. In actual practice we should also have

```
script
  state active:
      at_trigger see_salary:
            type_of (sender) = employee, /*is the sender employee?*/
            select (E in employees where E = sender),
            sender = E, /*is sender this employee?*/
            E: salary to message: param, /*return the salary*/
            next (active).
      at_trigger change_salary:
            type_of (sender) = employee, /*is the sender employee?*/
            select (E in employees where E = message: empl ),
            sender = E:mgr, /*is sender this employee's mgr?*/
            N from message: param, /*get new salary from message*/
            N > E: salary. /*is new salary higher?*/
            N to E: salary, /*assign to salary attribute*/
            next (active).
endscript .
```

Figure 2: A script for empl_adm.

provided this specification with some code for appropriate error messages. [2]

Messages must have a type defined as follows:

```
type message type1 is a message type
      has_a        empl: employee /*identifier of the employee*/
      has a        param: int /*for giving through salaries*/
```

From this example one can see that quite complex checks can be specified before some operation is carried out. By providing the low-level identity checks, discussed above, it is thus possible to create a truly safe knowledge base system in which the operations are carried out only when the appropriate authorization checks have been performed successfully.

As another example, take the situation that a salary can only be changed when both the manager and the boss are acting in concert. Suppose the boss must first send a message to the Empl adm and within 2 time units the manager must then send the change salary message.To provide for a secure checking the Empl adm gets an extra state: attention, which it gets when the boss sends that message. Only in this state Empl adm is willing to listen to the message change salary. After 2 units of time the Empl adm changes automatically in the state active, in which it is not willing to react on the manager's message. The script now runs as follows:

```
script
  state active:
      at_trigger see_salary: /*same as above*/
```

[2]Note that above and in the rest of this paper we allow ourselves some freedom as far as MOKUM syntax is concerned. Actually, sender is an attribute of message and getting values in and out of attributes has to be done in MOKUM more clumsily: one should write: "S **from** message: sender, S = E" instead of " sender = E".

```
            next (active).
            at_trigger change_to_attention:
            sender = boss,/*is the sender the boss?*/
            now+2 to time_trigger,
            next (attention).
  state attention:
      at_trigger change_salary: /*same as above*/
            next (active).
            at_time time_trigger:/*waited long enough*/
            next (active).
endscript
```

We have shown that quite complex contextual security rules can be put in the script part of a keeper of a collection.

Turning to the list of Constraints in section 3, we can easily see that all kinds can be dealt with. Take C4: when there are more than one collections involved with one keeper, then a keeper type can be specified in which these collections are values so that this keeper can be made responsible for keeping all the constraints. When there are more than one keepers involved, these have to communicate with each other. As an illustration, take as example that there are two empl_adm's, one for the male and one for the female employees. Let us call them M and F. The constraint is that their collections may not overlap. M and F have the same type: empl_adm. In this specification of Empl_adm it is not possible to refer to M and F directly, i.e. by names denoting the respective objects. It is, however, possible to specify an attribute which denotes the "other" keeper. Let this attribute be called "otherEA". When M and F are created this attribute gets as value F and M, respectively. The definition of empl_adm may now look like:

```
type empl adm
    private
      has a        otherEA:empl adm
      has a        nr of employees:int
      has a        employees: collection of employee
   script
     state active:
       at trigger add employee:
            send (otherEA, check membership, message),
            R from message:param,
            (R = 1, /*empl is member of the other EA*/ ;
            R = 0, /*empl is not member of the other EA*/
            message:empl into employees),
            next (active).
       at trigger check membership:
            sender = otherEA, /*only the other EA is entitled to get an answer*/
            (select (E in employees where E = message:empl),
            1 to message:param;
            0 to message:param),
            next (active).
   endscript .
```

It would be interesting to see whether a compiler can be made who generates code like this when given the constraint in its original form:

$$M : employees \cap F : employees = \emptyset.$$

Looking now back at the list of possible operations we remark that the relevant operations in MOKUM, such as **new** , **add_type** , **from** , **to** , **into** , **out of** , **destroy** (for types of objects) and **delete** (objects), all are used in the script part and thus under the control of the ICS designer.

4.2 Security in MOKUM

Let us first see how a typical security system can be implemented in MOKUM. In the next subsection some theoretical remarks will be given. Take the military situation as sketched in section 3.2. Suppose the objects of interest are documents characterized as top-secret (T), secret (S), confidential (C) and non-confidential (N) and that they are to be inspected by generals (G), lieutenants (L) and soldiers (S). Gs may see documents characterized S, C and N, Ls may see C and N documents and Ss may see only N documents. The way to implement a safe and secure system in MOKUM is to put the documents in a collection, and to install a keeper of that collection. The keeper is called Security Officer. The documents have an attribute sensitivity with possible values: 3, 2, 1 and 0.

Another attribute of a document is: contents, in which the contents of the document is put. Both attributes are supposed to be private. So, only Security Officers can manipulate both attributes. Military persons have a type which is a subtype of person, with (at least) one extra private attribute: clearance, with possible values: 3, 2 and 1. It is supposed that the particular object in charge of determining the clearance of a mil person is mpm, having type mil_person_mgr.

A possible definition of the types is given in figure 3.

```
type  mil person is a  person
    private
      has a      clearance: int
type  document is a  thing
      has a      identification: string
    private
      has a      sensitivity: int
      has_a      contents: string
type  mil person mgr is a  mil person
    private
      has a      mil persons: collection of  mil person
type  security_officer is a  mil person
    private
      has_a      documents: collection of  document
```

Figure 3: Type specification of security problem

In the script part of the security_officer the important security checking can be specified, as in figure 4.

```
script
  state active:
      at_trigger inspect_document:
            type_of (sender) = mil_person,
            select (D in documents where D:identification = message:id),
            new (M),add_type (M,message_type2,[(param1,sender)]),
            send (mpm,ask_for_clearance,M),
            /*the clearance is returned in param2 of the message M*/
            Clearance from M:param2, destroy (M),
            Clearance >= D:sensitivity,
            /*the test is successful*/
            D: contents to message: param;
            /*the test fails and no information is disclosed*/
            next (active).
endscript .
```

Figure 4: A script for the security problem.

As can be seen from this example, it is very important to have a close connection between the internal person objects and the real persons. It is considered part of the man-machine interface to make this connection. In a future project we will extend MOKUM such that a real person can do almost all that can be described in a script. So real protection is necessary then.

In [Olivier & von Solms, 1994] the authors formulate a taxonomy for security in O-O databases. We notice that many of the different systems they describe can be implemented in MOKUM, using the notion of private attributes and keepers of collections, as we have demonstrated above.

4.3 Visibility and accessability in MOKUM

In this subsection we derive some theory about visibility and accessibility in a MOKUM program. A MOKUM program consists of types, attributes, procedures and scripts. In procedures and scripts one can specify the use of types, object identifiers and attributes. In the rest of this section we shall use the term 'script' meaning both scripts and procedures.

MOKUM does not provide any protection on the usage of object identifiers and types. If somewhere in a script the value of an object identifier is known, e.g. because it is the parameter of a message, it can be used, by asking its type (**type of**) and its attributes can be manipulated, provided the attributes are available. Attributes declared private are protected in the MOKUM system. All other attributes can be seen and changed by all objects in the script of any type and no protection is provided.

Also, to make things very simple, protection is full, i.e. it includes read and write protection. In a future MOKUM system we will build in a more differentiated form of protection.

In the following we shall focus our attention on protection of private attributes. We will define visibility and accessibility.

A MOKUM program consists of:

- a set of (user-defined) types, called type_set;

- a set of attributes, called attr_set;
- a set of facts of the form: attr(T,A,CT) where T in type_set, A in attr_set and CT has the following form: [Case, Type], where Case = simple, or coll and Type is either elementary i.e. int, real or string, or is user defined: Type in type_set;
- a set of facts is a(T,S), where T and S in type_set, defining the usual generalization/specialization. The resulting graph must be cycle free. The meaning is that T inherits all attributes from S, in particular the private ones;
- a set of facts of the form: private(A), for some A in ATTR.

In a MOKUM program the attributes must be unique.

We now define visibility: Non-private attributes are visible in all types; for private attributes, the type in which the attribute is defined has evidently the property that that attribute is visible. Moreover, an attribute is visible within a type when that attribute is visible in a super type or when that type is the type of a collection keeper, whose collection has elements with a type in which that attribute is visible. In Prolog, this is defined very precisely as follows:

```
vis(A,T):- A in attr_set, T in type_set, not private(A).
vis(A,T):- A in attr_set, T in type_set, private(A),
    (attr(T,A,_);
      (attr(T,_,[coll,S]); is_a(T,S)), vis(A,S)
     ).
```

Visibility is a property which can easily be detected by the compiler using the Prolog rule given above. Accessibility presupposes visibility, i.e. the compiler has checked that access to an attribute is allowed. The MOKUM system also needs to check that the proper object is involved. This is evidently necessary. Just a check on visibility means that in the case where salary is a private object of employee, an employee can change his manager's salary when manager is an attribute of employee.

For proper access protection MOKUM applies the following rule: suppose the access involves an attribute of object O. The caller object CO must be the same as O, or must be one of the keepers of O. So:

```
acc(CO,O,A,T):- not private(A), vis(A,T).
acc(CO,O,A,T):- private(A), vis(A,T), (CO=O; keeper(CO,O)).
```

A keeper K of an object O of type t is an object of type kt and attr(kt,a,[coll,t]) is in the program, while indeed O has been put into K:a. Evidently, the latter can only be checked at run time.

Visibility is a necessary property to check before access to a certain operation by some object can be allowed. One can see from a simple example why a run time check on the object's identity is not sufficient.

Suppose in the script part of a person the attribute salary is mentioned, e.g. in a statement to change it. Now salary is supposed to be a private attribute of employee, to be manipulated only by an object of type employee or the keeper of employees. Without the visibility protection it would be possible for this person to change his/her salary because evidently the object identity test succeeds. A similar counter example where a keeper of a collection is involved is the following: a collection consists of persons, maybe a subtype of person, say tennis_player, and the keeper of this collection wants to know the salary of

these persons, which evidently is against the rules. Also in this case a simple object identifier test would be insufficient.

Visibility can be checked by the compiler, as has been remarked above, it can also be checked at run time (actually the representation of the MOKUM type specification, as generated by the compiler, looks quite similar to the above representation). The reason we let the compiler do this is of course efficiency: instead of checking visibility every time a private attribute is accessed this checking is done once by the compiler. When we have extended MOKUM with a facility that the "real" person can manipulate his internal counterpart person object as if a script were executed, the run time checking of visibility will be built in for this case.

Let us now see the protection theory applied to the example above. We have:

```
type_set = [person, mil_person, document, mil_person_mgr,
    security_officer]

attr_set = [name, clearance, sensitivity, mil_persons, documents]
        private(clearance).
        private(sensitivity).
        private(mil_persons).
        private(documents).
        is_a(mil_person, person).
        is_a(mil_person_mgr, mil_person).
        is_a(security_officer, mil_person).
        attr(mil_person,  clearance, [simple,int]).
        attr(document, sensitivity, [simple,int]).
        attr(document, contents, [simple,string]).
        attr(mil_person_mgr, mil_persons, [coll, mil_person]).
        attr(security_officer, documents, [coll, document]).
```

We shall now compute, using the Prolog system the visibility of name, clearance and sensitivity, by asking:

```
    vis(name,T)?-
person, mil_person, document, mil_person_mgr,
security_officer.
    vis(clearance,T)?-
mil_person, mil_person_mgr (2*), security_officer.
    vis(sensitivity,T)?-
document,  security_officer.
```

and vice versa asking what is visible by objects of type security_officer, mil person mgr, mil person and person:

```
    vis(A,security_officer)?-
         name, clearance, sensitvity, documents.
    vis(A,mil_person_mgr)?-
          name, clearance (2*),  mil_persons.
    vis(A,mil_person)?-
         name, clearance.
    vis(A,person)?-
         name.
```

For another interesting approach to visibility and in particular authorization in an O-O environment see [Rabitti, Bertino, Kim & Woelk, 91].

5 Constraints as MOKUM restrictions

There is a major problem attached to simply putting the maintenance of the constraints in the script part, namely the constraints have to be coded by the MOKUM programmer in the form of rather detailed MOKUM instructions. It would be nice that the MOKUM compiler takes care of some of these constraints. In this section we will see how this is done and how far we can go.

We will introduce the notion of restriction. A restriction is always connected to a type and to some attributes of that type. A restriction is furthermore inheritable. Both the compiler and the MOKUM kernel have to deal with a restriction. In principle, the compiler translates a restriction into Prolog code, while the kernel is performing an update after checking that the particular Prolog code is successfully executed. If not, the kernel refuses to carry out the update. An example of a restriction is the following one where a person's wage is related to that person's age:

```
type  person
        has_a          name: string
        has_a          wage: int
        has_a          age: int
        restriction    age, wage: Restr_proc1
    proc
      Restr_proc1:-
                A from age,
                W from wage,
                W < A.
    endproc
```

Each time age or wage gets a new value (operation OP1.4) this restriction is invoked substituting the new value into either age or wage, the Prolog procedure Restr_proc1 is invoked and it must return successfully.

With these kind of restrictions we are able to treat the constraints in categories C1.1 and C1.2 of section 3. Denoting the old value of an attribute, that is the value before the update, with A **from** attribute, and the new value with the attribute itself, we can relate old and new values of attributes, e.g. in the restriction:

```
        restriction  age: Restr_proc2
    proc
      Restr_proc2:-
                N = age, /*new value*/
                O from age, /*old value*/
                N > O.
    endproc
```

Such restrictions are appropriate for connection with one attribute as only one attribute can be changed at a time. These kinds of restrictions presuppose

that there always is an old value, which sometimes is not the case (Operation OP1.3). In MOKUM one can handle such cases by first checking whether the value is available, as is illustrated in the following example:

```
Restr_proc3:
        N = age,
        (O from age, !, N > O ;
        true /*there is no old value for age*/).
```

Constraints of category C2 can principally not be treated this way, they must be handled using a script. Such constraints typically refer to more than one object, such as the constraint that a spouse of a spouse of someone must be that someone. Evidently, such a constraint can not be maintained if the registration of a marriage involves two actions: changing the marital status of the husband and doing the same with the wife. This typical constraint must be defined in a script of the keeper of marriages.

Let us now turn our attention to the constraints of category C3. When the properties of the members are not involved the constraint usually refers to properties of the set itself, e.g. number of elements. The collection keeper should now have an extra attribute to keep track of such a property. Upon insertion and deletion of an element of the collection such an attribute must be updated and possibly checked. With the tools described above one can have:

```
type empl adm
     has a        nr_of_employees:int
     has a        max_nr_of_employees:int
     has a        employees: collection of employee
     restriction  nr_of_employees : Restr_proc4
  proc
    Restr proc4:-
          N from nr of employees, M from max_nr_of_employees,
          N < M.
  endproc
  script
          ...
      at trigger insert empl: ...
          N from nr of employees, N1 = N+1,
          N1 to nr_of_employees,
          /*if successful then insert in employee collection*/
          ...into employees, ...
          ...
  endscript
```

The constraints of category C3.2, where properties of the members of a collection are involved, can also be treated by restrictions. Take as example:

$\forall_{e \in employees}$: e.age > 20

This constraint requires that the update of the age attribute of some person involves a check whether that person is, as an employee, member of employees. Note that the MOKUM kernel knows the keepers of the collections to which a certain object belongs; this facility had to be implemented for proper access control anyhow. The above constraint is specified in the type of the collection keeper, in our example: empl_adm, as if it were a restriction on the elements of the collection:

restriction employees : Restr_proc5
proc forall employees: Restr_proc5:- (A **from** age, A > 20).
endproc

The compiler can see that this restriction is actually a restriction on age of employee objects and translates this restriction in the following one:

proc Restr_proc6:- (A **from** age, A > 20). **endproc**

This restriction is then connected to the age attribute in person (age in employee is inherited from person). The attribute age then has two restrictions connected to it: Restr proc1 and Restr_proc6. When the age attribute is changed Restr_proc1 is called always and Restr_proc6 is called only when the object is a member of an employees collection.

Also, when an element is added to the collection the restriction is executed of course.

Constraints involving existential quantifiers cannot be accommodated by restrictions in MOKUM. They should be treated in the script parts. The same holds true for constraints of category C3.3.

Looking at the list of operations, we see that the restrictions can be combined with the operations OP1.3, OP1.4 and OP1.5 (there is no difference between ordinary values and object-values in MOKUM) and OP3.1 and OP3.2 (the set-membership operations). For the other operations MOKUM restrictions cannot be used.

For security constraints, involving the context, in the form of the invoker and its environment, as introduced in section 3.2, we see that restrictions could be used, in combination with a MOKUM procedure. In the procedure we could specify constraints on invoker and environment. However, this would be very unnatural. A much simpler way is to specify the security constraints in the script, as we have shown at the end of section 4.

6 Conclusions

In this paper we have demonstrated how integrity constraints, referring in particular to the quality of the data, and security constraints, referring to protection and access rights, are treated in an integrated way in the MOKUM system. It is argued that all constraints, whether they refer to attribute values of one or several objects, whether they refer to properties of several collections, combined with properties of their members, or whether they refer to access rights, they can all be written as Prolog predicates in the form of a script. Many of them can also be specified in the form of restrictions on attributes and collections. These are much easier to read and to specify. For a secure implementation it turned out to be necessary to build in some low level control performed by

the MOKUM kernel, this pertained to the checking of restrictions as well as to access control.

One can compare this treatment with the one in [Bassiliades & Gray, 94], in which the authors describe how in their CoLan system all constraints are translated into Prolog predicates, which are also connected to certain update operations. No mention was made in their paper, however, of security and protection rules.

It is our conviction that the fact that in MOKUM has only two levels: objects and types, specifying constraints on objects and on collections can be done in a very direct way. It is also possible to make a compiler which takes some of the burden of the constraint specifier away. It would be nice to see how far we can go in building a compiler which translates the constraints written in a high level language, like FOL, into the scripts we use for MOKUM objects.

Another future addition to the MOKUM system is that the animation facility is extended with a window in which a real person, as end user, can interactively communicate with his "own" object. The manipulation available for the real person comprises all the commands which can be used in the trigger part of a script. Having this facility makes it possible to experiment with a knowledge base who is willfully changed in unforeseen ways in order to demonstrate that its constraints and security rules are sufficiently strong.

Another interesting problem is to make a tool which can check whether constraints are inconsistent. For example a constraint on an object type may be in conflict with a constraint on the elements of a collection of objects of that same type. This is a problem which in general leads to undecidability, but for special cases, for example, range restrictions, is solvable. A similar problem can be formulated for constraints concerning types and subtypes. These problems are to be worked out in the future.

Our group is also working on how to use linguistics in the area of Modelling Information and Communication Systems. One of the projects is how to translate a high level language, in which natural language and language in the form of pictures and diagrams is mixed, into MOKUM specifications. This involves also the security and integrity constraints we discussed in this paper.

References

[Bassiliades & Gray, 94] N.Bassiliades, P.M.D.Gray, CoLan: A Functional Constraint Language and i ts Implementation, Data and Knowledge Engineering, forthcoming.

[Croshere, vandeRiet & Blom, 93] R. Croshere, R.P. van de Riet, A.Blom, An Animation Facility to Simulate an Information and Communication System, in: C.Rolland (Ed.) Proceedings CAiSE93, Springer, 199 3, pp. 547-568.

[deJonge, 83] Wiebren de Jonge, Compromising Statistical Databases Responding to Queries about Means, ACM Transactions on Database Systems, Vol.8, No. 1, (March 1983) pp. 60-80.

[deJonge, 85] W.de Jonge, Security and Privacy in Information Systems, Some Technical Aspects, Ph.D. Thesis, Vrije Universiteit, Amsterdam, 1985.

[Grefen & Apers, 93] P.W.P.J. Grefen, P.M.G. Apers, Integrity control in Relational Database Systems - A n Overview, Data and Knowledge Engineering, Vol. 10, nr. 2 (March 1993), pp. 187-2 23.

[Kersten, 85] M.L. Kersten, A Model for Secure Programming Environment, Ph.D. Thesis, Vrije Universiteit, Amsterdam, 1985.

[Motschnig & Storey, 93] R.Motschnig-Pitrik, V.C.Storey, Grouping: an effective construct for mod elling sets , private communication, 1993.

[Olivier & vonSolms, 94] Martin S. Olivier, Sebastiaan von Solms, A Taxonomy for secure Object-Oriented Databases, ACM Transactions on Database Systems, Vol.19, No. 1, (March 1994) pp. 3-46.

[Paton, Diaz & Barja, 93] N.W.Paton, O.Diaz, M.J. Barja, Combining active rules and metaclasses for enhanced extensibility in object-oriented systems, Data and Knowledge Engineering, Vol. 1 0, nr. 1 (February 93), pp. 45-64.

[Rabitti, Bertino, Kim & Woelk, 91] Fausto Rabitti, Elisa Bertino, Won Kim, Darrell Woelk, A Model of Authorization for Next-Generation Database Systems, ACM Transactions on Database Systems, Vol.16, No. 1, (March 1991) pp. 88-131.

[Sicherman, deJonge & vandeRiet, 83] George L.Sicherman, Wiebren de Jonge, Reind van de Riet, Answering Queries Without Revealing Secrets, ACM Transactions on Database Systems, Vol.8, No. 1, (March 1983) pp. 41-59.

[Stonebraker, 76] Michael Stonebraker, Eugene Wong, Peter Kreps, The Design and Implementation of INGRES, ACM Transactions on Database Systems, Vol.1, No. 3, (Sep. 1976) pp. 189-222.

[vande Riet,Kersten & Wasserman, 82] R.P. van de Riet, M.L. Kersten, A.I. Wasserman, A Module Definition Facility for Access Control in Communicating Data Base Systems, in: vandeRiet & Litwin [Eds.] Distributed Data Sharing Systems, North Holland Publishing Company, 1982. Also published in: Proceedings of a Conference on Security and Privacy, 1980, IEEE Computer Society.

[vande Riet, Kersten,deJonge & Wasserman, 83] R.P. van de Riet, M.L. Kersten, W. de Jonge, A.I. Wasserman, Privacy and Security in Information Systems using Programming Language Features, Information Systems, Vol. 8, 2, 1983, pp. 95-103.

[vandeRiet, 89] R.P. van de Riet, MOKUM: An object-oriented active knowledge base system , Data and Knowledge Engineering, Vol 4, No. 1, North Holland, 1989, pp 21-42.

[vandeRiet & Gamito, 90] R.P. van de Riet, M.V.Gamito Dignum, Interface MOKUM/Ingres IR-215, Faculteit Wiskunde en Informatica, Vrije Universiteit, Amsterdam, mei 1990.

[Wasserman, vandeRiet, Kersten & Leveson, 83] Anthony I. Wasserman, Reind P. van de Riet, Martin L. Kersten, Nancy Leveson, A Formal Integrated Approach to Data and Usage Integrity in Health Information Systems, in: G.Griesser,J.P Jardel, D.J.Kenny, K.Sauter (Eds.), Data Protection in Health Information Systems, Elsevier-Science Publishers, 1983.

Constraint Management in Knowledge Bases, in: R.P. van de Riet, Kennisbanken, Course notes, Faculteit Wiskunde en Informatica, Vrije Universiteit, Amsterdam, 1993, pp. 153-178.

Queries, updates, and transactions:

Chair: O. Costich

Consult. to NRL, VA

Field Level Classification and SQL

Simon R. Wiseman

Defence Research Agency, Malvern, Worcestershire WR14 3PS, England

ABSTRACT

The SWORD secure DBMS is unique in that it provides field level classifications which are not equivalent to row level classifications. This has a significant impact upon the query language used to access the database. In particular, it is necessary to handle the results of expressions which clients are able to know exist, but are not cleared to know the actual value. Also, it is desirable to generate detailed, field level information labels. The paper focusses on the effect of these requirements on the semantics of SWORD's secure variant of SQL.

1. INTRODUCTION

In the SWORD Secure Relational DBMS [Wood *et al* 92], Information Flow is controlled using field level classifications. This is in contrast to the emerging Secure RDBMS products, which only add a single classification per row. The important difference between the two approaches is that with field level classification a client is able to detect the existence of data even when their clearance is insufficient to allow them to observe the data's value. With such row level classification, the only data that can be detected is that which can also be observed.

This paper describes the extended form of SQL, called Secure SQL (SSQL), which has been devised as the query language for SWORD. The emphasis is on the semantic problems associated with field level classifications, support for trustworthy clients and providing detailed Information Labels.

The addition of row level classifications to a DBMS need have very little impact on the query language. The essential addition is the ability to use the row label in expressions. With SWORD's field level classifications there is a need to handle values which the client is *not cleared* to observe, and this makes a significant difference to the semantics of the query language.

Alternative field level classification schemes have been proposed, but these are equivalent to row level classifications [Qian&Lunt92]. In these schemes, the existence of a field is hidden from a client with insufficient clearance to observe its contents, by the presence of a lowly classified field (which, if nothing else, contains a null). This results in polyinstantiation, with consequent loss of system integrity [Wiseman89]. SWORD deliberately adopted the Insert Low approach [Wiseman90] to avoid polyinstantiation and the associated problems.

Support for trustworthy clients is often provided by a system of privileges, but these can be rather crude and difficult to manage. Instead SSQL allows the text of a query and the fact that it is issued to be classified lower than the client's clearance, while still preserving the Information Flow policy. Such support is easier to use and manage but significantly complicates the Information Flow constraints inherent in the semantics, though the paper does not describe these in any detail.

The Secure DBMS products all allow simple Information Labels to be attached to rows retrieved from the database. SSQL tries to provide more detailed information, by

This work was carried out for the UK Ministry of Defence under order NNR/19h/93.

labelling both retrieved rows and their fields, and by taking into account the subtleties of complex predicates. DBMS Information Labels, which are sometimes called Advisory labels, are strongly related to CMW [Berger *et al* 90] "Information Labels" although the mechanics of their operation are quite different.

2. SECURITY POLICY

The SSQL security policy constrains information flows and defines Information Labels.

2.1 Information Flow Security

The Information Flow Security policy is a strong statement about the way in which information is permitted to flow through the DBMS. All inputs (queries) are labelled with classifications that indicate the sensitivity of the information conveyed by the input. All clients are labelled with a classification, their clearance, which restricts the information they are permitted to observe.

Roughly, the policy, which is called "No Flows Down", is that no client may learn anything about information entered into the database, unless their clearance dominates the classification given that information. This can be put more formally as follows:

> Clients that have insufficient clearance to see any difference between two sequences of inputs, may not see any difference in the sequence of results caused by those inputs.

This is a generalisation of the Non Interference property defined by Goguen and Meseguer's seminal paper [Goguen&Meseguer82].

2.2 Labelling Results

The result of a query provides some basic facts directly, but implies more facts by virtue of the context in which it arises. That is, knowing the answer gives some information, but knowing the question as well gives much more. Generally, the direct facts are classified lower than the implied facts.

In some applications, it appears useful to know what other clients, in terms of clearance, are able to learn the various facts encoded in a result. Thus results are labelled to provide this information. A formal statement of the policy is given in [Wiseman93].

3. MULTI-LEVEL QUERIES AND RESULTS

In SSQL, tables, queries and the results of a select query are all multi-level entities.

3.1 Multi-level tables

Tables in SSQL are multi-level entities, structured in two-dimensions: row and column. This two-dimensional structure can be represented as an object hierarchy consisting of a number of rows each with the same number of fields.

Each table has a classification, which is used to protect the table's schema information. The existence of a table is classified independently by the classifications in the hierarchical directory system which is used to name tables [Cant *et al* 94]. The directory system is an important extension to SQL which allows tables to be named in a

similar way to files in an operating system. The directory structure allows tables with sensitive names to be hidden inside highly classified directories.

The existence of a row in a table is given a classification, though this row existence classification must dominate the classification of the table. In addition, no two rows in a table are allowed to have incomparable existence classifications. This constraint permits the rows to be stored in order of increasing existence classification, which is necessary to avoid highly classified information from being encoded into the time taken to process a query.

The existence of each column in a table is given a classification, with similar restrictions. However, unlike with hidden rows, the data in hidden columns can be processed without affecting query execution time through the judicious use of list manipulations. Thus rows can be inserted or deleted, without execution time revealing highly classified information, even if the table contains hidden columns.

Each field in a table is also given a classification. This field classification must dominate both the row existence class of the row it is within and the column existence class of the column it is within. That is, the object hierarchy which makes up a table is 'Compatible' in the [Bell&LaPadula76] sense.

For each column in a table, there is a row in a data dictionary table which describes it. If the existence of a column is to be hidden from users with low clearances, then the corresponding row in the data dictionary must also be hidden. This is achieved by classifying the existence of the row to the classification of the column it describes. This is the only practical use of tables with varying row classifications that has so far been identified, so the restrictions on their use does not appear to be a problem.

3.2 Multi-level queries

Queries in SSQL are generally multi-level entities. However, those clients which cannot be trusted to maintain the separation between the different elements of the query, are constrained to issuing single-level queires. Multi-level queries allow trustworthy clients to change the database in ways which are visible to clients with lesser clearance, while at the same time preserving "No Flows Down".

Within the SSQL syntax, it is only possible to label expressions. This is achieved by enclosing the expression in brackets and prepending a (constant) classification. The classification of the query as a whole is given by a classification prepended to the query.

The following query is shown as an example. It is given an overall classification of Confidential, but part of its where clause is classified higher, at Secret.

```
[C] SELECT * FROM flights
                    WHERE dest IS NOT NULL AND [S]( cargo = 'Bombs' )
```

This query can be seen in its entirety by clients with clearances of Secret or above. However, clients with Confidential clearance would see a censored structure:

```
[C] SELECT * FROM flights
                    WHERE dest IS NOT NULL AND [S]( not cleared )
```

In practice, one client is not able to observe the queries issues by another. However, it is sometimes possible to infer details about those queries by observing the effect they have on the database. In these cases, SSQL is such that effects observed by a client will not

reveal anything about the nature of parts of others' queries that are classified higher than the client's clearance.

In the case of stored procedures, a client may execute a query which is not completely visible to them. In this case, the query will fail if that part of the query turns out to be relevant to the query's execution. For example, suppose a client cleared to Confidential executes the above query. This will successfully return no rows if all rows in the table have a null dest field, however it will fail if any row has a non-null dest, because it is then necessary to compute the 'invisible' condition.

3.3 Multi-level results

The result of an update, insert or delete query is a simple message, however select queries yield Derived Tables which are themselves multi-level entities. The structure of a Derived Table is similar to that of real tables, but the labels are used to convey additional information about how the result was formed. These labels may be 'Incompatible', meaning that the labels may decrease as the tree is descended. Hence, untrusted clients may usefully be given a multi-level result, but their clearance will be greater than all the labels within it. The derivation of the labels in Derived Tables is discussed in §6.

4. CALCULATING THE VALUE OF EXPRESSIONS

In SSQL, the client does not always have sufficient clearance to be able to compute an expression. This has a significant effect on the semantics of expressions.

4.1 Arithmetic and Comparisons

An SSQL query will only be evaluated in terms of those rows whose existence is classified at a level which is dominated by the client's clearance. In general, the query will have to compute expressions for these rows. These expressions are used in Where clauses, Select lists and so forth.

In some cases, the results of these expressions will depend upon the contents of fields which are classified at a level which is not dominated by the client's clearance. This means the value of the result cannot be revealed to the client, even though the client is cleared to know that there is such a result.

In effect, the result of computing an expression for a given row is either a value of the appropriate type, or a special value called *Not Cleared*. If the client's clearance permits them to learn the result of the expression, they receive the proper value, while if their clearance is insufficient they receive *Not Cleared*. Note that *Not Cleared* is special only because it cannot be stored in a field of a table, even though it can be stored by the client software if required.

The simplest form of expression is a constant. The result of such an expression is that constant, and is always visible to the client because the client provided the constant as part of the query.

Another simple form of expression is a column name, the result of which is the value of the row's field in that column. However, in SSQL the client's clearance is first checked against the field's classification. If the clearance dominates the classification, the result is the field's contents, otherwise the true value is ignored and the result is *Not Cleared*.

For arithmetic and simple comparisons, the result is *Not Cleared* if any of the arguments are *Not Cleared*. Some examples are shown in the following table:

>	1	2	*Not Cleared*
1	False	False	*Not Cleared*
2	True	False	*Not Cleared*
Not Cleared	*Not Cleared*	*Not Cleared*	*Not Cleared*

Actually, SSQL is slightly over-strong in this respect because it is sometimes possible to give a correct answer even when one argument is *Not Cleared*. For example, the smallest integer is less than or equal to all other such integers of the same precision. Thus a query language with the following behaviour for 16 bit two's complement integers would still be secure:

>	-32768	2	*Not Cleared*
-32768	False	False	False
2	True	False	*Not Cleared*
Not Cleared	*Not Cleared*	*Not Cleared*	*Not Cleared*

However, SSQL does not provide for such special cases as any implementation would be rather inefficient and the relaxation does not seem particularly worthwhile.

4.2 Field & Row Labels

In SSQL it is possible to use the labels attached to fields and rows as expressions of type Class. If the rows are from a joined Derived Table, it is possible to select the row labels of the individual tables that contribute to the join. The syntax is as follows:

```
CLASS OF column name          -- label of a field
CLASS OF ROW                  -- label of row
CLASS OF ROW OF table name    -- label of contribution to joined row
```

A client who has sufficient clearance to detect the existence of a row, is always able to observe the row's existence classification and the classifications of its fields. Although some applications may not wish this information to be so freely available, it is not precluded by "No Flows Down". When necessary, stricter control can be achieved through the judicious use of triggers [Lewis *et al* 92].

The emerging secure DBMS products only support row labels, and all essentially treat this as an additional column of the table. Unfortunately, this approach does not scale to field level labelling and so SSQL uses a different form of concrete syntax. Another advantage of the SSQL syntax, is that it requires no additional reserved words. This is because, in SQL, "<ident> OF" is not legal syntax.

4.3 AND/OR Predicates

Boolean expressions, which are called predicates in standard SQL, can often be computed accurately even when one argument is *Not Cleared*. For example, when False is ANDed with either True or False, the result is False. So if the client's clearance is sufficient to determine that one argument is False, the result can be determined without examination of the other argument, even if it is *Not Cleared*.

One further complication in SQL is that truth values are three-valued, since a predicate may evaluate to Null (called *unknown* in the standard). Thus in SSQL, truth values are four-valued, since predicates evaluate to False, True, Null or *Not Cleared*. This is shown in the following truth tables:

AND	False	True	Null	*Not Cleared*
False	False	False	False	False
True	False	True	Null	*Not Cleared*
Null	False	Null	Null	*Not Cleared*
Not Cleared	False	*Not Cleared*	*Not Cleared*	*Not Cleared*

OR	False	True	Null	*Not Cleared*
False	False	True	False	*Not Cleared*
True	True	True	True	True
Null	False	True	Null	*Not Cleared*
Not Cleared	*Not Cleared*	True	*Not Cleared*	*Not Cleared*

The behaviour of AND and OR predicates with respect to *Not Cleared* values provides some very important functionality. When updating a table, the Where clause expression is used to determine whether a particular row should be modified (see §5). However, if computing the Where clause results in *Not Cleared* for any row, the query is abandoned with no rows being modified. If the client wishes to update those rows which meet the Where clause criteria, except where the clearance is insufficient to determine this, it is necessary to augment the Where clause as follows:

UPDATE ..... WHERE *condition* AND *clearance is high enough to compute condition*

As a concrete example, consider the following table and update query issued by a client with a clearance of Confidential:

Payload

Id		Weight	
123	[U]	42	[C]
456	[U]	42	[S]
789	[U]	0	[C]

```
UPDATE Payload SET Id = 0
    WHERE Weight > 10 AND CLEARANCE DOM CLASS OF Weight
```

With the first row, the client is able to observe the Weight and hence can determine that the Where clause expression is True. Similarly, with the third row, the client is able to determine that the Where clause is False. However, with the second row, the client is not able to observe the Weight, hence the first part of the Where clause is *Not Cleared*, but the second part of the Where clause yields False. Thus the entire clause is False and the row is not selected for update. Without this second part, the Where clause would yield *Not Cleared* and the whole update would fail.

Such Where clauses are particularly useful, but adding the correct clearance check is tiresome and error prone, particularly when AND and OR predicates are nested. So, to make things easy, SSQL introduces two new monadic boolean operators (predicates), called DEFINITELY and POSSIBLY. These are defined by the following truth table:

x	DEFINITELY x	POSSIBLY x
False	False	False
True	True	True
Null	Null	Null
Not Cleared	False	True

The example query given above can now be restated using the DEFINITELY operator:

```
UPDATE Payload SET Id = 0 WHERE DEFINITELY Weight > 10
```

The POSSIBLY operator would be used when some action is required if a condition can be seen to be True or the client's clearance is insufficient to compute it

4.4 Set Functions

SQL provides a variety of set functions (sometimes called aggregate functions) such as Sum and Maximum. These are applied to a multi-set of values which are obtained by evaluating an expression for a number of rows. Optionally, duplicates may be removed from the multi-set before the function is computed. In SSQL additional set functions are provided, including set forms of Least Upper Bound and Greatest Lower Bound on classifications.

For SSQL, the result of one of these set functions is *Not Cleared* if any of the values in the multi-set is *Not Cleared*.

Standard SQL also includes an additional set function called COUNT. This has two forms, COUNT(*) which counts the number of rows in the set being considered, and COUNT(DISTINCT *exp*) which counts the number of rows that give a distinct, non null answer for the given expression.

For SSQL, the COUNT(*) function is straightforward, in that it simply provides a count of the number of rows in the set being considered. The set will only ever contain rows whose existence is classified with a classification that is dominated by the client's clearance. Thus the client is always cleared to detect all the rows and hence may always learn their number.

In SSQL the Distinct form of Count is more interesting because it includes an implied equality test and test for Null. If any rows are such that the expression yields *Not Cleared*, it is not possible to determine whether the row should be discounted, either on the grounds of giving Null or being a duplicate value. So, if the client's clearance is insufficient to compute the expression for any of the rows being considered, the overall result of the Count Distinct is *Not Cleared*.

The following table shows the results of computing various set functions against the rows in the Payload table for clients with two different clearances:

CLIENT CLEARANCE	SUM(Weight)	COUNT(*)	COUNT(DISTINCT Weight)
{C}	*Not Cleared*	3	*Not Cleared*
{S}	84	3	42

4.5 Quantified Predicates

In standard SQL, a Quantified Predicate is one where an expression is compared against each of the values in a one column wide Derived Table. All the usual arithmetic comparisons may be made and there are two forms of the predicate; one which gives whether some selected value meets the condition and one which gives whether they all do.

In effect, such predicates are a series of tests, ANDed together in the case of the ALL form and ORed together in the case of the SOME form. Hence in SSQL the rules of AND and OR with respect to *Not Cleared* values applies.

4.6 Exceptions

The SQL standard does not detail how arithmetic exceptions such as overflow should be handled. In a secure environment, such exceptional behaviour could be the source of illicit information flows, so it is important to be clear about the effects.

The following fragment of an embedded SSQL/Ada program shows how inappropriate exceptional behaviour could be exploited by a client to obtain the (integer) value of a field which has a classification not dominated by the client's clearance:

```
declare
    SSQL DECLARE STATEMENT st AS
        SELECT Weight + :P FROM Payload WHERE Id = 456;
    SSQL DECLARE CURSOR c;
    SSQL DECLARE VARIABLE res : Integer;
    SSQL DECLARE VARIABLE try : Integer := Integer'last; -- start with largest
begin
    loop
        SSQL OPEN CURSOR c FOR st USING try FOR p
        begin
            SSQL FETCH FROM c INTO res
            exit; -- weight + try = integer'last
        exception
            when ssql_interface.overflow => null; -- weight + try > integer'last
        end;
        try := try - 1;
    end loop
    text_io.put_line( "weight = " & integer'image(integer'last-try) );
end
```

This repeatedly attempts to add some constant to the quantity field of a particular row. The constant starts off as large as possible, and is gradually decreased. At first an arithmetic overflow will occur during query processing, but this is handled by the loop and the search continues. Eventually the constant is made small enough so that an overflow does not occur. At this point a forced exit from the loop is made and the value in the field can be computed.

SSQL avoids such problems by introducing exceptions as special values. With respect to classification checks, these exceptions are treated like any other data. In effect, the clearance check occurs before the arithmetic takes place The following table, which gives some example results for addition of 16 bit twos complement integers, shows this:

+	-32768	1	32767	*Not Cleared*
-32768	*Overflow*	-32767	-1	*Not Cleared*
1	-32767	2	*Overflow*	*Not Cleared*
32767	-1	*Overflow*	*Overflow*	*Not Cleared*
Not Cleared	*Not Cleared*	*Not Cleared*	*Not Cleared*	*Not Cleared*

For comparisons the result is an exceptional value if any of the arguments are exceptional values, but for ANDs and ORs an exceptional value as an argument need not lead to an exceptional result, as the following table illustrates:

x	y	x AND y	x OR y
True	Exception	Exception	True
False	Exception	False	Exception
Not Cleared	Exception	*Not Cleared*	*Not Cleared*

5. WHERE, GROUPBY AND HAVING CLAUSES

An SQL query contains three clauses, Where, Groupby and Having, which are used to identify the rows that are to be selected, updated or deleted.

The Where clause is processed first. It consists of a predicate (boolean expression) which is evaluated for each of the rows in the table being processed. If the expression yields False or Null for a particular row, that row is removed from future consideration. The default Where clause is the constant True.

The Groupby clause, which consists of a set of column names, is now applied. The rows being processed are divided into the minimum number of groups, such that the rows in a particular group all have the same values in the fields of the specified columns. If no Groupby clause is specified, all the rows are formed into one large group.

Finally, the Having clause is applied. This is a predicate which is evaluated for each group. The expression must evaluate to the same value for each row in a group, otherwise the query fails completely. Any group for which the Having clause is False or Null is now discarded *en masse*. The default Having clause is the constant True.

The main effect of the Groupby clause is on set functions. For a particular row, a set function applies to all the rows in that row's group. In SSQL, Set functions in the Where clause apply to all the rows in the table being processed.

In SSQL, for Update and Delete queries and Select queries that occur inside other queries, the client clearance must be sufficient to compute the value of the Where clause for each and every row in the table. If the result of the Where expression for any row is *Not Cleared*, or is some exception, the query fails. For Select queries at the outer level, such rows are ignored and the select continues, although the client is able to ask whether this has occurred.

Processing the Groupby clause is in effect a number of equality tests. Thus the client must be cleared to observe all the data in the columns mentioned in the Groupby list. If this were not so for some row, it would not be possible to decide in which group the row belongs. Hence, if the classification of any of the data in the Groupby columns is not dominated by the client's clearance, the query fails.

SSQL also allows rows to be grouped according to the classification of fields in a column. However, a client is always able to observe the field classifications in any row that they are cleared to know exists. Hence it is always possible to form groups on this basis. The following example illustrates how the syntax of Groupby clauses has been extended to allow this:

SELECT AVG(weight) FROM Payload GROUP BY CLASS OF Weight

The rules for processing the Having clause are similar to those for the Where clause.

6. DERIVED TABLES

A Derived Table can be formed by extracting data from a stored table or by combining other Derived Tables in a variety of ways.

6.1 Field Classifications

The field classifications of a Derived Table are effectively Information Labels, which are the minimum clearance a client can possess such that it is still possible, using some query or other, to learn the facts derived from the fields.

The Information Label given for a constant is always Unclassified (lattice bottom). This is because all clients know about all constants, so anyone can construct a constant expression.

For a column name expression, the field's classification is used as the Information Label. This is because the field classification is the lowest clearance a client can have if they are to learn the field's value.

The Information Label generated for arithmetic operations and comparisons is given by the least upper bound of the arguments' Information Labels. This is because a client must be cleared to know the value of all arguments in order to learn the result.

Deriving the Information Label generated for an AND/OR predicate is more complex. This is because the result of such an expression can sometimes be known even when some arguments are not known. As the following tables show, the Information Label is usually the least upper bound of the arguments' Information Labels. However, if the result of an AND is False, then knowing just one of the False arguments is enough to learn the answer. Hence the Information Label should be that of the 'lowest' False argument. Similarly for OR.

AND		OR	
False	'lowest' False argument	False	lub arguments
True	lub arguments	True	'lowest' True argument
Null	lub arguments	Null	lub arguments
Not Cleared	lub arguments	*Not Cleared*	lub arguments

Unfortunately, security classifications are not always comparable, so there may not be a 'lowest'. In the case where the Information Labels of all the False arguments are comparable, the Information Label of the AND's result is simply the greatest lower bound of the False arguments' Information Labels. When they are incomparable, there is no one correct Information Label. This is because there are several possible 'minimum' clearances which are sufficient to learn the result.

Rather than support multiple Information Labels, which is just too difficult to implement and does not seem very useful, the incomparable Information Labels are combined in the following way. First the labels are partitioned into the minimum number of sets of labels which are comparable. The greatest lower bound from each set is taken and then the least upper bound of these is used as the result's Information Label. Although this operation sounds complex, an iterative bit-pattern implementation is quite straightforward.

For arithmetic set functions, such as SUM and AVG, the Information Label is the least upper bound of all the Information Labels of the expressions calculated for each row.

The set function COUNT(*) returns the number of rows in the table (strictly, the current group of the table). The result is completely unrelated to the contents of the rows, but reveals information about the existence of each row. Hence, the Information Label is the least upper bound of the existence classifications of each row.

For COUNT(DISTINCT exp) the expression is calculated for each row, and rows giving duplicate or null values are ignored. Thus the result depends on both the values of the expressions and the existence of the rows. Hence, the Information Label is the least upper bound of the Information Labels for the expressions and the existence classification of each row (note that for Derived Tables, the row existence classification may strictly dominate the expression's Information Label).

For quantified predicates, the Information Label is derived according to the rules given for AND/OR predicates, though the existence classification of the rows is also taken into account.

6.2 Row Classifications

When a Derived Table is described by a Select query, the presence of a row is determined by the query's Where clause. Thus the row's Information Label is a classification which is the minimum clearance needed to discover that the Where clause expression is True. This classification may be higher than that needed to compute the selected expressions. Thus, the Information Label attached to a row need not be dominated by the Information Labels on the fields.

In general, the row's existence is also influenced by the Groupby and Having clauses and the row classification is actually the least upper bound of the classifications needed to evaluate these clauses as well.

7. SUMMARY

This paper has described Secure SQL, a variant of standard SQL89 which has been extended to support the field level classifications provided by the SWORD Secure Relational DBMS. The language also provides detailed Information Labels and allows trustworthy clients to work freely within the bounds of their clearance while still enforcing "No Flows Down".

8. REFERENCES

Secure Computer System: Unified Exposition and Multics Interpretation
D.E.Bell & L.J.LaPadula
ESD-TR-75-306 (MITRE Report MTR-2997), January 1976.

Compartmented Mode Workstation: Prototype Highlights
Jeffrey L. Berger, Jeffrey Picciotto, John P.L. Woodward & Paul T. Cummings
IEEE Trans. on Software Engineering, Vol 16, Num 6, June 1990, pp608-618.

The SWORD Data Dictionary
Christopher Cant, Sharon Lewis & Simon Wiseman
DRA Technical Rerport 94002, February 1994

Security Policies and Security Models
J.A.Goguen & J.Meseguer
Procs. IEEE Symp. on Security and Privacy, Oakland CA, April 1982, pp11-20.

Providing Security in a Phone Book Database Using Triggers
S.R.Lewis, S.R.Wiseman & N.D.Poulter
Procs 8th Computer Security Applications Conference, San Antonio, Texas, November 1992, pp 85-96

Tuple-level vs. Element-level Classification
Xiaolei Qian & Teresa Lunt
Procs. IFIP WG11.3 6th Annual Working Conference on Database Security, Vancouver, British Columbia, August 1992

On the Problem of Security in Data Bases
Simon R. Wiseman
Procs. IFIP WG11.3 3rd Annual Working Conference on Database Security, Monterey, CA, September 1989

Control of Confidentiality in Databases
Simon Wiseman
Computers & Security Journal, Vol 9, Num 6, October 1990, pp529-537

Security Properties of the SWORD Secure DBMS Design
Simon Wiseman
Procs. IFIP WG11.3 7th Annual Working Conference on Database Security
Huntsville AL, September 1993

The SWORD Multilevel Secure DBMS
A.W.Wood, S.R.Lewis & S.R.Wiseman
Defence Research Agency Report 92005, February 1992.

Degrees of Isolation, Concurrency Control Protocols, and Commit Protocols

Vijayalakshmi Atluri [a], Elisa Bertino [b,*], and Sushil Jajodia [a]

[a]Center for Secure Information Systems and Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444, U.S.A.

[b]Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy

Abstract

In this paper, we make two contributions related to transaction management in multilevel secure distributed databases. First, we present a secure locking protocol (SLP) that provides different degrees—0, 1, 2, and 3 - of isolation. Next, we present a secure early prepare commit protocol (SEP) that not only preserves atomicity of distributed transactions, but can be integrated with SLP without violating the isolation requirements as well. Both SLP and SEP take advantage of the isolation degree of the transaction being executed. For degrees 0, 1, and 2, SLP is free of starvation, and SEP requires only 4n messages (where n is the number of sites participating in the commit protocol). For degree 3 isolation, SLP may suffer from starvation, although the probability of starvation is quite small; and SEP may sometimes require more than 4n, but never more than 6n messages. We suggest a way to reducing this additional cost in messages using synchronized clocks.

Keyword Codes: D.4.6; H.2.0; H.2.4
Keywords: Security and Protection; Information Systems, General; Systems; Multilevel Security; Atomic Commit, Isolation, Concurrency Control; Serializability

1. INTRODUCTION

In a distributed database (DDB), there are several logical objects, which are physically located at different sites or nodes. A distributed transaction, though initiates at one site, may require to access objects stored at remote sites. To guarantee correct executions of distributed transactions, each site in the DDB is equipped with a concurrency control protocol and a commit protocol. There are several different concurrency and commit protocols with two-phase locking (2PL) and basic two-phase commit (2PC) as being the most well-known concurrency and commit protocols, respectively.

Although most conventional (single-level) commercial systems use locking based mechanisms for concurrency control, they do not always use 2PL. They offer different *degrees*

*The work of E. Bertino was carried out while visiting George Mason University during summer 1993.

of isolation—0, 1, 2, and 3—to transactions [GLPT76, GR93]; each transaction has the option of specifying the degree of isolation required.† The reason for providing different options is that transactions that specify lower degree of isolation require fewer and shorter duration locks, leading to improvement in the amount of concurrency and response time.

Moreover, most commercial systems (e.g., IBM's LU6.2 and Digital's DECdtm) use the early prepare commit protocol (EP) rather than 2PC as the commit protocol. This is because these systems do not support request/response paradigm, thus an implementation of 2PC in these systems requires 6n messages (where n is the number of sites participating in the commit protocol) compared to 4n messages required by EP.

In keeping with the existing practices in commercial systems, we first offer in this paper a secure (distributed) locking protocol (SLP) that provides all four degrees of isolation. A nice property of SLP is that it is not subject to starvation for degree 0, 1, or 2 isolation. Although SLP may suffer from starvation for degree 3 isolation, we show that the probability of starvation is quite small. The significance of this result is that locking, which is the universally accepted mechanism for concurrency control in conventional databases, can be used in multilevel secure databases as well, provided we are willing to tolerate a small probability of starvation.

Next, we give a secure analog of EP, called SEP, that not only preserves atomicity, but requires only 4n messages for degree 0, 1, or 2 isolation. For degree 3 isolation, it may sometimes require more than 4n messages, but it never requires more than 6n messages. We suggest a way to reducing this additional cost in messages using synchronized clocks. We also show that SLP and SEP can be easily integrated to provide the desired degree of isolation as well.

This paper is organized as follows. We begin in section 2 with a review of related work. In section 3, we present our distributed multilevel secure (MLS) DDB model. In section 4, we describe the lock-based protocol, SLP, which provides different degrees of isolation. Since SLP is susceptible to starvation for degree 3 isolation, subsection 4.1 contains a probability model showing that the probability of starvation is quite small. In section 5, we describe SEP for different degrees of isolation. Since SEP sometimes requires extra rounds of messages for degree 3 isolation, section 6 describes an optimization of SEP, called O3SEP, which reduces this extra number of messages. Finally, section 7 presents conclusions.

2. RELATED WORK

Although most of the research in MLS transaction management has focused on centralized databases, there has been some recent activity that deals with DDBs. In [JM93, JMB93], Jajodia, McCollum, and Blaustein study the secure analogs of 2PL and commit protocols. They modify 2PL to give a secure 2PL protocol (S2PL) that yields degree 3 isolation. S2PL, like our SLP, is susceptible to starvation.

[JMB93] also shows how EP, 2PC, and some optimizations of 2PC (e.g., presumed commit) can be modified to be secure. While their modifications to 2PC and presumed commit yield a protocol that can be integrated with S2PL without any violation to global

†Degree 0 is also known as *chaos*, degree 1 as *browse*, degree 2 as *cursor stability*, and degree 3 as *repeatable reads* or *serializable*.

consistency their modifications to EP cannot guarantee the global consistency of the distributed data when used in conjunction with S2PL. In this paper, we give a different modification to EP that ensures global consistency of data.

3. THE DISTRIBUTED SYSTEM MODEL

A distributed system consists of a set of nodes N, where each node $N_i \in N$ is an MLS DBMS. The various nodes in the distributed system are connected via communication links. We assume that these communication links are tamper proof. Each node is supported by a Trusted Computing Base (TCB), which is responsible for mediating all database accesses and cannot be bypassed. Each node also has its own local transaction manager (LTM) and distributed transaction manager (DTM). The DTM acts as an interface between the LTM and distributed transactions (those that originate at the site or are remote).

3.1. The MLS DBMS model

We model the MLS DBMS as a quadruple $< D, T, S, L >$, where D is the set of data items (*objects*), T is the set of transactions (*subjects*), S is the partially ordered set of access classes (or security levels) with an ordering relation $\leq$, and L is a mapping from $D \cup T$ to S. For every $x \in D$, $L(x) \in S$, and for every $T_j \in T$, $L(T_j) \in S$. In other words, every data item as well as every transaction has a security class associated with it.

We extend the mapping L such that it maps each MLS DBMS N_i to an ordered pair of security classes $L_{min}(N_i)$ and $L_{max}(N_i)$. Clearly, it should always be the case that $L_{min}(N_i) \leq L_{max}(N_i)$ and $L_{max}(N_i), L_{min}(N_i) \in S$. In other words, every MLS DBMS in the distributed system has a range of security levels associated with it. For every data item x stored in an MLS DBMS N_i, $L_{min}(N_i) \leq L(x) \leq L_{max}(N_i)$. Similarly, for every transaction T_j executed at N_i, $L_{min}(N_i) \leq L(T_j) \leq L_{max}(N_i)$. A node N_i is allowed to communicate with another node N_j only if $L_{max}(N_i) = L_{max}(N_j)$. The reader may refer to [JM93] for additional details on distributed MLS DBMS model.

Our security policy is based on the Bell-LaPadula model [BL76]. According to this model, the following two conditions are *necessary* for a system to be secure:

- A transaction T_j is allowed to read a data element x only if $L(x) \leq L(T_j)$.
- A transaction T_j is allowed to write a data element x only if $L(x) = L(T_j)$.

The second property, which allows transactions to write only at its level, is a restricted version of the $\star$-property. The original $\star$-property proposed in the Bell-LaPadula model allows transactions to write into levels above their security level. However, in the database context, it seems prudent to disallow transactions that write to higher levels [JK90].

In addition to these two requirements, a secure system must guard against illegal information flows through signaling and covert channels.

3.2. The distributed transaction model

When a transaction T_i is submitted to the DTM of a local MLS DBMS, if T_i is a local transaction, DTM simply passes it to the LTM. When T_i is distributed, DTM assumes the role of the coordinator and, thus, is responsible for resolving references to the data

items accessed by that transaction so as to determine where these data items are located. DTM then generates the subordinate processes (from now on we refer to these as simply subordinates). A distributed transaction T_i can originate at an MLS DBMS N_k where $N_k \in N$ if $L_{min}(N_k) \leq L(T_i) \leq L_{max}(N_k)$. A subordinate $T_{i,k}$ can execute at N_k if $L_{min}(N_k) \leq L(T_{i,k}) \leq L_{max}(N_k)$.[‡]

Each subordinate $T_{i,k}$ at node N_k inherits the same security level as that of T_i: $L(T_{i,k}) = L(T_i)$. Even if a subordinate is a read-only transaction that reads data items below the security level of the originating transaction, it is executed as a read-down transaction at the corresponding subordinate node, its clearance level being same as that of the originating transaction. Every subordinate $T_{i,k}$ is subjected to the security restrictions described in the earlier subsection. These when combined with the range constraints result in the following conditions on subordinates. A subordinate $T_{i,k}$ can execute at node N_k only if

- whenever $T_{i,k}$ wishes to read an item x, $L_{min}(N_k) \leq L(x) \leq L(T_i)$, and
- whenever $T_{i,k}$ wishes to write an item x, $L(x) = L(T_i) \leq L_{max}(N_k)$.

3.3. The transaction model

We model a *transaction* T_i (either a local transaction or a subordinate) as a sequence of read and write operations on data items. We use $r_i[x]$ and $w_i[x]$ to denote the read and write operations issued by a transaction T_i on a data item x.

Sometimes transactions acquire a lock before they read or write an item. They acquire a share lock (S-lock) for reading and an exclusive lock (X-lock) for writing an item. All locks acquired by a transaction must eventually be released.

Definition 1 Two locks $o_i[x]$ and $o_j[x]$ are *compatible* if $i = j$ or neither of them is an X-lock. □

An S-lock is compatible with other S-locks on an item and, therefore, multiple transactions can hold an S-lock on an item at the same time. However, only one transaction can hold an X-lock on an item at any given time.

Definition 2 A transaction is *well-formed with respect to writes* if it acquires an X-lock before writing a data item. A transaction is *well-formed* if it acquires an S-lock (X-lock) before reading (writing) a data item. □

Definition 3 A transaction is *two-phase with respect to writes* if it does not acquire any X-lock after it releases an X-lock on a data item. A transaction is *two-phase* if it does not acquire any more locks once it releases a lock on a data item. □

3.4. Degrees of Isolation

We present below the definitions for degree 0, 1, 2, and 3 isolation proposed by Gray et al. in [GLPT76].

[‡]We use $T_{i,k}$ to denote the subordinate of T_i at node N_k.

Definition 4

Degree 0: A transaction observes degree 0 isolation if it is well-formed with respect to writes.

Degree 1: A transaction observes degree 1 isolation if it is well-formed with respect to writes and also two-phase with respect to writes.

Degree 2: A transaction observes degree 2 isolation if it is well-formed and two-phase with respect to writes.

Degree 3: A transaction observes degree 3 isolation if it is well-formed and two-phase.

□

4. SECURE LOCKING PROTOCOL FOR DIFFERENT DEGREES OF ISOLATION

For the distributed executions of transactions to be correct, it is necessary to ensure atomicity, consistency, isolation, and durability [GR93]. Isolation means that the system gives every transaction the illusion it is being executed alone all by itself, although other transactions are run concurrently in the system.

While total isolation is desirable, most commercial systems offer different degrees of isolation, viz., degree 0, 1, 2 and 3. In SQL2, a transaction can specify its desired degree of isolation, declaring the degree of sharing it can tolerate. Lower degrees of isolation improve the performance of the system, though achieved at the expense of consistency.

Intuitively, with degree 0 isolation, a transaction is not allowed to update a data item while another transaction is updating it. With degree 1 isolation, a transaction cannot overwrite a data item until the completion of the transaction which wrote it earlier. With degree 2 isolation, a transaction can read data items only from committed transactions. Degree 3 provides complete isolation to transactions. By choosing this degree of isolation, a transaction must wait to either read or write a data item for the completion of all other transactions that read or wrote it.

The stringent requirements imposed by multilevel security dictate modification of the locking protocols given in the previous chapter. An ideal locking protocol in a multilevel secure database management system must possess the following key properties:

- Provide the desired degree of isolation for transactions
- Preserve security (i.e., it must obey Bell-LaPadula restrictions and be free of signaling channels)
- Be implementable with untrusted code
- Be free of starvation. Starvation may occur because high level transactions may be subjected to indefinite delays or suspended repeatedly by low level transactions to prevent signaling channels.

Since transactions can specify their desired degree of isolation, in this section, we propose a secure locking protocol that gives degree 0, 1, 2, and 3 isolation. This protocol is a modified version of the protocol given in [GLPT76, GR93] so that it meets the security requirements.

Algorithm 1 [Secure Locking Protocol (SLP)]

- For degree 0 isolation,
  - Whenever a transaction or a subordinate wishes to write a data item x, it must first acquire an X-lock on x before writing x.
  - A transaction or a subordinate T_i releases the X-lock on a data item x when the write operation $w_i[x]$ is completed.
- For degree 1 isolation,
  - Whenever a transaction or a subordinate wishes to write a data item x, it must first acquire an X-lock on x before writing x.
  - A transaction or a subordinate T_i releases all X-locks only when it commits or aborts.
  - A transaction or a subordinate cannot acquire any more X-locks once it releases an X-lock.
- For degree 2 isolation,
  - All transactions and subordinates are well-formed. That is, whenever a transaction or a subordinate wishes to read (write) a data item x, it must first acquire an S-lock (X-lock) before reading (writing) x.
  - A transaction or a subordinate T_i releases all X-locks only when it commits or aborts. However, T_i must release an S-lock as soon as the corresponding read operation is completed.
  - A transaction or a subordinate cannot acquire any more X-locks once it releases an X-lock.
- For degree 3 isolation,
  - All transactions and subordinates are well-formed. That is, whenever a transaction or a subordinate wishes to read (write) a data item x, it must first acquire an S-lock (X-lock) before reading (writing) x.
  - A transaction or a subordinate T_i releases all locks only when it commits or aborts. However, T_i must release an S-lock on a data item x whenever another transaction T_j requests an X-lock on x such that $L(T_j) < L(T_i)$. In such an event, T_i is aborted.
  - A transaction or a subordinate cannot acquire any more locks once it releases a lock.

□

It is important to note that SLP is single-level, and therefore, can be implemented with untrusted code. However, since the SLPs at all security levels use a common lock-table, the lock-manager must be trusted.

For degree 2 isolation, whenever a low transaction requests an X-lock on a data item while a high transaction's read operation on the same data item is being executed, the high transaction is not required to release its S-lock to accommodate the write request by the low transaction. This is because we assume that the read operation is instantaneous and thus does not cause any delay in processing the low transaction's write request, and therefore, does not introduce a signaling channel. In other words, we make the assumption that the three actions—acquiring an S-lock, reading the data item and then releasing the S-lock—are executed instantaneously. This is not an unreasonable assumption since SLP for degree 2 isolation does not require the S-locks to be two-phase; the S-locks are short duration locks and are released as soon as the read operation is performed.

On the other hand, for degree 3 isolation, whenever a low transaction requests an X-lock on a data item on which a high transaction already has an S-lock,[§] the high transaction releases its S-lock and thereby allows the low transaction to proceed with its write operation. Otherwise, the lower level transaction would need to wait for the release of this S-lock by the high transaction. This situation can be exploited by two colluding transactions at levels high and low to establish a signaling channel. To prevent such illicit flow of information, a secure system must prioritize lower level transactions over their higher level counterparts while allocating the locks. In this process, some transactions may get aborted repeatedly, resulting in starvation.

To reduce the amount of starvation, Jajodia and McDermott [MJ92] propose a variation of this approach. According to this variation, whenever a high transaction prematurely releases its S-lock on a low data item due to security reasons, it does not abort or roll-back entirely. The high transaction continues to hold its X-locks on high data items, marks the low data item in its private workspace as unread and retries reading this data item by entering into a queue. This queue maintains the list of all high transactions waiting to reread that particular data item, and enables the first transaction in the queue to be serviced first. Though transactions are not two-phase, this approach guarantees serializability. We refer the reader to [MJ92] for additional details and for the proof of correctness.

Theorem 1 If a transaction observes the SLP, then any legal history[¶] H will give that transaction degree 1, 2, or 3 isolation, as long as other transactions in H are at least degree 1.

Proof: Proof is similar to the one given in [GLPT76, pages 384–386]. □

4.1. Probability of starvation

A very simple model, adapted from [GR93], is provided to estimate the probability of starvation. Suppose that the database has R data items. Moreover, suppose that

[§]Recall that high transactions are allowed to read, but not write data at lower levels. Therefore, a high transaction will never be able to acquire an X-lock on a lower level data item.

[¶]A history is said to be *legal* when no two incompatible locks on an item are simultaneously held by transactions in that history.

there is a high transaction that wishes to read l low data items and that there are n low transactions, each modifying r low data items. These transactions are all running concurrently.

The probability that the high transaction must release a lock on a low data item, because a low transaction is modifying it, is approximately $P = (nr)/R$. Note that, as pointed out in [GR93], $nr << R$, i.e., most data items are unlocked most of the time. The probability that the high transaction is aborted because one of its locks on the low data items has been released early is given by:

$$\begin{aligned} TA &= 1 - (1-P)^l \\ &= 1 - (1 - \binom{l}{1} P + \binom{l}{2} P^2 + \quad + (-1)^r \binom{l}{l} P^l) \\ &\approx lP \\ &= \frac{lnr}{R} \end{aligned}$$

The high-order terms can be dropped because $nr << R$ and, therefore, $P << 1$.

The probability that the high transaction is aborted again, after being restarted the first time, is given by TA^2. Therefore, the probability that a transaction is aborted n times, due to early release of locks, is given by

$$TA^n = (\frac{lnr}{R})^n \qquad (\star)$$

Since in most cases the number of data items locked by the transactions is a small subset of the total data items, expression $(\star)$ rapidly decreases with the increase in n. As an example, consider the case of a database containing 1,000,000 data items, with 100 low transactions, each modifying 100 data items. Moreover, consider a high transaction that reads 25 low data items (i.e. 25% of the data modified by a low transactions). The probability that the transaction is aborted once is 1/4, and the probability that it is aborted twice is 1/16, which is already a quite small probability.

5. SECURE EARLY PREPARE PROTOCOL FOR DIFFERENT DEGREES OF ISOLATION

In this section, we present SEP that takes into consideration different degrees of isolation. We assume that SLP is being used as the concurrency control algorithm by all LTMs.

Algorithm 2 [Secure Early Prepare (SEP)]

When a user who is logged on at security level s initiates a distributed transaction T_i at a node N_j, the user must specify T_i's degree of isolation. The DTM at N_j acts as the coordinator for T_i, and initiates the first phase of SEP.

The prepare phase:

- The coordinator generates subordinates $T_{i,1}, T_{i,2}, \ldots, T_{i,n}$[‖] and sends[**] them to the nodes $N_1, N_2, \ldots, N_n$, respectively. The coordinator also sends the security level, which is s, and the isolation degree, which is same as the isolation degree specified for T_i by the user, with each subordinate.

- The DTM at each node N_k, $k = 1, \ldots, n$, hands $T_{i,k}$, its security level, and its degree of isolation to LTM. $T_{i,k}$ is executed by the LTM, taking into consideration the isolation level of $T_{i,k}$.

  - For degree 0 isolation, LTM acquires an X-lock for each item before it is written by $T_{i,k}$. These locks are *short* locks meaning that they can be released as soon the write has taken place. DTM sends a **yes** vote to the coordinator if $T_{i,k}$ successfully completes its execution; it sends a **no** vote otherwise.
  - For degree 1 isolation, LTM acquires an X-lock for each item before it is written by $T_{i,k}$. These locks are *long* locks meaning that they must be held until $T_{i,k}$ commits at the end of the decision phase. DTM sends a **yes** or **no** vote to the coordinator depending on whether or not $T_{i,k}$ completes successfully.
  - For degree 2 isolation, LTM acquires an S-Lock (X-lock) on an item before it is read (written) by $T_{i,k}$. S-locks are short locks, while X-locks are long. DTM sends a **yes** or **no** vote to the coordinator depending on whether or not $T_{i,k}$ completes successfully.
  - For degree 3 isolation, LTM acquires an S-Lock (X-lock) on an item before it is read (written) by $T_{i,k}$. S-locks as well as X-locks are long locks. If $T_{i,k}$ completes successfully, DTM augments its **yes** vote to the coordinator with a *read-low* indicator. A one-bit *read-low* indicator is added whenever $T_{i,k}$ has read an item from a lower level. DTM sends a **no** vote if LTM cannot commit $T_{i,k}$.

  A subordinate that sends an yes vote to commit is said to be in a *prepared state.*

The decision phase:

- Suppose the coordinator receives **yes** votes from all its subordinates.

  - For degree 0, 1, or 2 isolation, the coordinator commits T_i and then sends **commit** messages to all its subordinates.
  - For degree 3 isolation, there are two cases. If no subordinate has read data from lower levels, the coordinator commits T_i and then sends **commit** messages to all its subordinates. On the other hand, an extra round of messages is required between the coordinate and all those subordinates N_j that sent the read-low indicator with their yes vote.

‖We assume T_i is decomposed into n subordinates, and $T_{i,j}$, the subordinate at the originating node N_j is one among them.

**We use bold letters to indicate messages.

* The coordinator sends to each N_j a **confirm** message to confirm the commit.
* If N_j has not released its S-locks on any lower level data item during the time it has been in the prepared state, it responds with a **confirmed** message; otherwise, it sends a **not-confirmed** message.
* If the coordinator receives a **confirmed** message from all N_j's to which the coordinator has sent the additional round of messages, then it sends **commit** messages to all its subordinates; otherwise it sends **abort**.

If the coordinator receives at least one no vote or if it times out waiting for a vote, it aborts the transaction, and sends **abort** messages to all subordinates.

- Each subordinate is committed or aborted according to the message received, and then an **acknowledgment** of this fact is sent back to the coordinator.
- After receiving the acknowledgment from all the subordinates, the coordinator terminates T_i. □

5.1. Discussion

If we compare the number of messages required by SEP to that required by EP, EP always requires about 4n messages (where n is the number of subordinates), while SEP requires 4n messages for degree 0, 1, and 2 isolation, but sometimes requires more than 4n messages for degree 3 isolation. An extra round of messages is required between the coordinator and those nodes where the subordinates have read data from the lower levels. This is because degree 3 isolation in the distributed setting requires that not only each subordinate must be two-phase, but the distributed transaction as a whole must be two-phase as well (see [JMB93, Lom93, MLO86]). A simple way to ensure this is to require that all S-locks be held until the commit of the transaction. Unfortunately, we cannot impose such a restriction in a multilevel secure environment; a subordinate must release its S-lock on a data item whenever a lower level transaction requests an X-lock on the same data item.

Note that this release of locks does not cause any violation of the degree 3 isolation requirements if it occurs while the subordinate is still being executed (i.e., before the subordinate enters the prepared state). In such a case, the subordinate can be either aborted or reexecuted. However, if the release of locks occurs while the subordinate is in the prepared state, the subordinate can be neither aborted nor started over again. We illustrate this further by way of an example.

Let T_1 and T_2 be two distributed transactions as follows:

$$\begin{aligned} T_1 &= r_1[x]r_1[y]w_1[z], \quad L(T_1) = high \\ T_2 &= w_2[x]w_2[y], \qquad\quad L(T_2) = low \end{aligned}$$

Suppose T_1 is initiated at node N_c, and T_2 is initiated at node N_b. Furthermore, assume that data item x is stored at node N_a, y is stored at N_b, and z is stored at N_c. The coordinators, N_c and N_b, generate the subordinates, and send them to the corresponding remote nodes. Accordingly, N_c divides T_1 into three subordinates, $T_{1,a}$, $T_{1,b}$ and $T_{1,c}$, and

then sends $T_{1,a}$ and $T_{1,b}$ to N_a and N_b, respectively. Similarly, N_b, upon dividing T_2 into two subordinates $T_{2,a}$ and $T_{2,b}$, sends $T_{2,a}$ to N_a.

The execution of these subordinates at each of the nodes may result in a distributed history D, as shown in figure 1. At N_a, the following sequence of events takes place: After successful execution, the subordinate $T_{1,a}$ votes yes and enters the prepared state. At this point, the low level subordinate $T_{2,a}$ arrives. $T_{1,a}$ releases its S-lock on x, enabling $T_{2,a}$ to acquire an X-lock on x. At N_b, the subordinate $T_{1,b}$ is successfully executed after the commit of $T_{2,b}$. At N_c, T_1 is committed after the coordinator receives the yes vote from all the subordinates.

	N_a	N_b coordinator of T_2	N_c coordinator of T_1
high	$r_1[x]$ prepared	$r_1[y]$	$w_1[z]$
	T_1 releases locks		
low	$w_2[x]$	$w_2[y]$	

Figure 1 The distributed history D

Clearly, the distributed history D is not serializable since T_1 is serialized before T_2 at N_a, while the serialization order is reversed at node N_b. A moment of reflection shows that this inconsistency arises because the distributed transaction T_1 is not two-phase, although it is well-formed.

It is easy to see that SEP manages to avoid the above problem. During the prepare phase of the protocol, $T_{1,a}$ and $T_{1,b}$ send read-low indicators with their yes votes. As a result, during the decision phase, coordinator N_c sends a confirm message to both N_a and N_b. Since $T_{1,a}$ has released its S-lock on x, N_a responds with a not-confirmed message, and N_b responds with a confirmed message since $T_{1,b}$ has not released its shared lock on the low data item y during the prepared state. Because N_c receives a not-confirmed message, it decides to abort T_1 and informs all the subordinates of the outcome. Thus, our protocol is able to avoid the problem.

5.2. Proof of correctness

Theorem 2 Suppose that every DTM uses SEP for atomic commitment of distributed transactions and that every LTM uses SLP for concurrency control. Then any legal history H consisting of local and distributed transactions will give transactions degree 1, 2, or 3 isolation, as long as other transactions are at least degree 1.

Proof: Suppose T_i is a transaction in H. We prove this theorem in both cases where T_i is a local transaction and T_i is a distributed transaction.

First, suppose that T_i is a local transaction in H. Then T_i is given degree 1, 2, or 3 isolation, as long as other transactions in H are at least degree 1. This is because, in such a case, this theorem reduces to theorem 1.

Next, suppose T_i is a distributed transaction in H initiated at node N_j. Assume that T_i requires to be executed at n nodes. The DTM at N_j generates $T_{i,1}, T_{i,2}, \ldots, T_{i,n}$ and sends them to their respective nodes $N_1, N_2, \ldots N_n$.

We prove this theorem in the following three cases.

Case 1: First we show that if T_i specifies its degree of isolation as degree 1, then it is given degree 1 isolation as long as other transactions in H are at least degree 1.

Every LTM at nodes $N_1, N_2 \ldots N_n$ observe the SLP for degree 1 isolation to schedule the operations of $T_{i,1}, T_{i,2} \ldots T_{i,n}$. In other words, every subordinate of T_i is well-formed with respect to writes and two-phase with respect to writes. According to definition 4, T_i is given degree 1 isolation.

Case 2: In this case, we show that if T_i specifies its degree of isolation as degree 2, then it is given degree 2 isolation as long as other transactions in H are at least degree 1.

Every LTM at nodes $N_1, N_2 \ldots N_n$ observe the SLP for degree 2 isolation to schedule the operations of $T_{i,1}, T_{i,2} \ldots T_{i,n}$. In other words, every subordinate of T_i is well-formed and two-phase with respect to writes. According to definition 4, T_i is given degree 2 isolation.

Case 3: In this case, we show that if T_i specifies its degree of isolation as degree 3, then it is given degree 3 isolation as long as other transactions in H are at least degree 1.

In this case, we need only to argue that every distributed transaction T_i observing SEP protocol for degree 3 isolation is always two-phase and, therefore, T_i sees degree 3 isolation in H.

According to SEP for degree 3 isolation, a transaction or subordinate must release an S-lock on a data item if some other lower level transaction or subordinate requests an X-lock on the same data item.

Suppose none of the subordinates of T_i release their S-locks. In such a case, we can trivially see that the distributed transaction T_i is well-formed and two-phase and therefore T_i is given degree 3 isolation

Now suppose that at least one of the subordinates of T_i reads a low data item x and releases its S-lock on x. If one of the subordinates of T_i, say $T_{i,k}$, releases the S-lock on x before entering the prepared state, then by voting **no** it chooses to abort T_i. On the other hand, suppose $T_{i,k}$ releases its S-lock on x during its prepared state. According to SEP, since the the DTM at N_j receives a read-low indicator, it sends an additional round of message to N_k. Since $T_{i,k}$ responds with a **not-confirmed** message, T_i is aborted. Therefore, whenever any of the subordinates of T_i releases its S-lock during the prepared state of the SEP protocol, T_i gets aborted. In other words, SEP allows a distributed transaction T_i to commit only if all its subordinates continue to hold all their respective locks until T_i's commit. Since T_i is well-formed and two-phase, according to definition 4, T_i is given degree 3 isolation. □

6. AN OPTIMIZED DEGREE 3 SECURE EARLY PREPARE (O3SEP)

For degree 3 transaction, SEP as described above has two drawbacks. In addition to sometimes requiring more than 4n messages, it is overly pessimistic. Any high subordinate that reads low data is aborted if any of its S-locks on the low data are broken while it waits for the confirm message from the coordinator. Thus, SEP aborts a transaction if

there is a possibility of a violation of the two-phase requirement. It is entirely possible that the transaction is two-phase, even though some of the S-locks are broken.

We can improve in both these areas if we assume that clocks in the distributed system are synchronized.[††] This is not an unreasonable assumption [Lis91]. Using time services such as network time protocol [Mil90] or Digital time service [Dig89], it is possible to have distributed clocks that are synchronized within a millisecond, "even after extended periods when synchronization to primary reference sources has been lost" [Mil90].

In this section, we propose an optimization to SEP for degree 3 isolation, called O3SEP, that uses the synchronized clocks. O3SEP uses the clocks to isolate the exact situation in which the two-phase requirements are violated. This optimization reduces not only the number of messages, but the number of transactions being aborted as well.

In the O3SEP protocol, we maintain for every transaction the time at which each lock has been granted. We also maintain the early lock release time for each transaction if the transaction prematurely releases its S-lock on a lower level data item in order to accommodate a lower level transaction. We present next the necessary notation that will be used in this protocol.

Notation: Given a subtransaction $T_{i,j}$, $t_{i,j}^{lock}$ denotes the time at which the last lock is obtained by $T_{i,j}$. Given a subtransaction $T_{i,j}$ that has read a low data and voted yes, $t_{i,j}^{vote}$ denotes the time when the yes reply is sent. Moreover, given a subtransaction $T_{i,j}$ that has read a low data and voted yes but later forced to prematurely release some locks, $t_{i,j}^{release}$ denotes the time of the first early release of lock has occurred.

Given a transaction T_i, let

- $max_{T_i}^{lock}$ = maximum$\{t_{i,j}^{lock} \mid T_{i,j}$ is a subtransaction of $T_i\}$.

- $min_{T_i}^{release}$ = minimum$\{t_{i,j}^{release} \mid T_{i,j}$ is a subtransaction of T_i, such that $T_{i,j}$ has read low data and has released some lock early$\}$.

In other words, $max_{T_i}^{lock}$ denotes the time of the latest lock acquired by transaction T_i, whereas $min_{T_i}^{release}$ denotes the time of the earliest lock release performed by a subordinates that has read low data. □

Algorithm 3 [Optimized Degree 3 Secure Early Prepare (O3SEP)]

When a user who is logged on at security level s initiates a distributed transaction T_i at a node N_j, the user must specify T_i's degree of isolation. The DTM at N_j acts as the coordinator for T_i, and initiates the first phase of O3SEP.

[††]Note that the improvement in the message cost is without considering the cost incurred in maintaining a synchronized clock.

The prepare phase:

- The coordinator generates the subordinates, $T_{i,1}, T_{i,2}, \ldots T_{i,n}$ and sends them to the nodes $N_1, N_2, \ldots N_n$, respectively. The coordinator also **sends** to each of the subordinate nodes the security level s of each $L(T_{i,k})$, $k = 1, 2, \ldots n$.

- The DTM at each N_k executes $T_{i,k}$ as a transaction of level s and responds with a vote as follows:

  It sends a **yes** vote and $t_{i,k}^{lock}$ if the subordinate successfully completes its execution, otherwise it sends a **no** vote.

  If the subordinate that votes **yes** has read a data item x such that $L(x) < L(T_{i,k})$, then the subordinate includes in its vote message, in addition to the vote, one bit *read-low($T_{i,k}$)* indicator (that indicates that it has read low data) and $t_{i,k}^{vote}$.

The decision phase:

- If the coordinator receives at least one **no** vote or it times out waiting for a vote, it aborts the transaction and sends **abort** message to all its subordinate nodes.

  The coordinator commits T_i if all subordinate nodes voted **yes** and the coordinator has not received *read-low($T_{i,k}$)* from any N_k. It then sends **commit** messages to all its subordinate nodes.

  If the coordinator receives a *read-low($T_{i,k}$)* from some N_j, then the following additional steps are performed by the coordinator:

  - determine $max_{T_i}^{lock}$
  - for each N_j such that $t_{i,j}^{vote} < max_{T_i}^{lock}$, an additional round of messages are sent to node N_j
    - The coordinator sends to node N_j, a **confirm** message to confirm the commit.
    - If node N_j has not released its read-locks on a lower level data item during the prepared state, then it responds with a **confirmed** message, otherwise, it sends a **not-confirmed** message together with $t_{i,j}^{release}$.
  - At this point, if the coordinator receives a confirmed message from all N_j to which the additional of messages have been sent, then it sends **commit** to all its subordinate nodes.

    Otherwise it evaluates $min_{T_i}^{release}$ and if $min_{T_i}^{release} > max_{T_i}^{lock}$, it sends **commit** to all its subordinate nodes, otherwise, it sends **abort**.

- The subordinate node either commits or aborts the subordinate according to the message received, and then it sends an **acknowledgment** back to the coordinator.

- After receiving the acknowledgment from all the subordinates, the coordinator terminates T_i.

Theorem 3 If all transactions observe degree 3 isolation, every DTM uses O3SEP (algorithm 3) for atomic commitment, and every LTM uses SLP (algorithm 1) for concurrency control then every legal history H consisting of local and distributed transactions will be serializable

Proof: To prove this theorem, it is enough to prove that every distributed transaction T_i in H is always two-phase. If none of T_i's subordinates do not release any locks to accommodate a write request from a lower level transaction or subordinate, then T_i obviously is two-phase since O3SEP allows T_i to release all its locks only at the commit time. Therefore, we now need to prove that even though T_i releases some of its locks during its execution, it is still two-phase. For T_i to be not two-phase, at least one of T_i's subordinates must acquire a lock after a subordinate of T_i releases its S-lock. In such a case, $min_{T_i}^{release} < max_{T_i}^{lock}$. According to the O3SEP protocol, transaction T_i will be aborted. This guarantees that every distributed transaction in H is two-phase. Therefore, H is serializable. □

7 CONCLUSION

In this paper, we have given a secure locking protocol (SLP) and a secure commit protocol (SEP) for different degrees of isolation. SLP is free of starvation, and SEP requires only 4n messages for degree 0, 1, and 2 isolation. For degree 3 isolation, SLP may suffer from starvation, although the probability of starvation is quite small, and SEP may sometimes require more than 4n, but never more than 6n messages. We suggest a way to reducing this additional cost in messages using synchronized clocks.

ACKNOWLEDGMENTS

We are indebted to Joe Giordano of Rome Laboratory and LouAnna Notargiacomo and John Vasak of The MITRE Corporation for making this work possible.

REFERENCES

[BL76] D.E. Bell and L.J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, The Mitre Corporation, Bedford, MA, March 1976.

[Dig89] Digital Equipment Corp. *Digital Time Service Functional Specification, Version T.1.0.5*, 1989.

[GLPT76] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. Granularity of locks and degrees of consistency in a shared data base. *Modelling in Data Base Management Systems*, pages 365–394, 1976.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, California, 1993.

[JK90] Sushil Jajodia and Boris Kogan. Transaction processing in multilevel-secure databases using replicated architecture. In *Proc. IEEE Symposium on Security and Privacy*, pages 360–368, Oakland, California, May 1990.

[JM93] Sushil Jajodia and Catherine D. McCollum. Using two-phase commit for crash recovery in federated multilevel secure database management systems.

In C. E. Landwehr et al., editor, *Dependable Computing and Fault Tolerant Systems, Vol. 8*, pages 365–381, New York, 1993. Springer-Verlag.

[JMB93] Sushil Jajodia, Catherine D. McCollum, and Barbara T. Blaustein. Integrating concurrency control and commit algorithms in distributed multilevel secure databases. In T. F. Keefe and C. E. Landwehr, editors, *Database Security VII: Status and Prospects*, pages 109–121. North Holland, 1993.

[Lis91] Barbara Liskov. Practical uses of synchronized clocks in distributed systems. In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, pages 1–9, August 1991.

[Lom93] David Lomet. Using timestamping to optimize two phase commit. In *Proc. of the PDIS Conference*, pages 48–55. San Diego, CA, January 1993.

[Mil90] David L. Mills. *Network time protocol (version 3) specification, implementation, analysis.* DARPA Networking Group Report, July 1990.

[MJ92] John McDermott and Sushil Jajodia. Orange locking: Channel-free database concurrency control via locking. In *Proc. of the 6th IFIP WG 11.3 Workshop on Database Security*, Vancouver, BC, August 1992.

[MLO86] C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the R* distributed database management system. *ACM Transactions on Database Systems*, 11(4):378–396, December 1986.

Status reports on current projects:

Chair: J. Biskup

Uni. Hildesheim, Germany

Trusted ONTOS Prototype
Preliminary Considerations

Marvin Schaefer
ARCA Systems, Inc.
Columbia, MD

Sandra Wade
ONTOS, Inc.
Vienna, VA

In the late summer of 1993, the authors[1] began research under contract to the National Security Agency and Rome Laboratory to begin development of an informal access control model [1] for a trusted object-oriented database management system (ODBMS). This study is intended to serve as the basis for future efforts to produce a trusted prototype of an ODBMS offering features comparable to those required for Class B1 of the DoD's Trusted Computer System Evaluation Criteria (TCSEC) and the associated Trusted Database Interpretation (TDI) of the TCSEC.

The philosophy behind object oriented technology is becoming the *de rigueur* standard for the industry, even though there is presently no universal model that serves as a standard for individual ODBMS implementations. Several ODBMS products are currently serving a growing user community. They are being used with greater frequency by the government and industry because they offer many benefits over existing technologies such as increased performance for complex applications, support for unusual data types, and a highly flexible data model. Additionally, with the reduction of budgets in both the government and industry, object-oriented technology is gaining a wider audience for its potential to reduce overall life cycle costs by enabling component based software development, promoting software re-use, and supporting extensible solutions. It is evident that ODBMS technology will be the basis for future DoD database applications. There is a clear need for a high integrity, multilevel secure, ODBMS.

Although there have been numerous paper studies, there are presently no *worked* examples of a trusted ODBMS, extant or under development. It is equally important to note that although the more traditional concepts and architecture of relational DBMS (RDBMS) tend to dominate the TDI, there are no *interpretations* of how specific TCSEC requirements are to be applied to an ODBMS. The present effort is intended to support future research and development needed in order better to understand a) the security related issues in the design and implementation and b) the evaluation, and especially the *assurance* requirements for a high-integrity, multilevel secure ODBMS that offers B1 features.

This study is intended to take a fresh look at the trusted DBMS problem. Previous, relational model-based approaches, have largely been based on a set of security architectures that lead to polyinstantiation or selective database replication as a means of preserving confidentiality. However, use of this strategy is often at the cost of database consistency, integrity, performance, and the ability to see updates without delay. Further, the semantics and operational consequences of polyinstantiation have sometimes proven to be inadequately understood by users and have resulted in database update inconsistencies. Given that object-oriented architectures invite the introduction of new security architectures, the opportunity is present to re-examine alternatives that could result in a more favorable tradeoff between the objectives of confidentiality and database integrity.

---

[1] Marv Schaefer was affiliated with CTA Incorporated at the time.

Panel: Perspectives on database security:

Chair: M. Morgenstern

Cornell Uni. NY

PANEL
Perspectives On Database Security

Panel chair: Matthew Morgenstern – Xerox/Cornell U., Ithaca NY, USA
Panelists: Joachim Biskup – U. Hildesheim, D
Klaus Dittrich – U. Zurich, CH
Carl Landwehr – Naval Research Lab., Washington, USA
Marv Schaefer – ARCA Systems, Maryland, USA

This panel will explore the *themes and trends in database security*, including security policies and models as well as the user's perspective and requirements.

As security for relational database systems matures, we see that the new commercially available products offer hooks for flexible security policies – to accommodate application-specific requirements. These hooks loosen the restrictions on information flow between levels in a controlled manner. For discretionary access controls, the analysis of group-based privileges and the emergence of new security paradigms, such as separation of powers, also suggests a trend toward choices among security policies through flexible configuration of the security parameters. This raises the question of whether there might be several orthogonal dimensions that help to define a space of alternative security policies and models, and the extent to which these dimensions can be made non-interfering.

The need for assurance and certifiability conflicts with flexible security policies. One must determine the consequences of each alternative security policy, and assure that for each combination no operational flaws or loopholes exist. Thus vendor products often seek certification based upon the one preferred configuration.

Another trend is the expansion of multilevel security from relational systems to encompass object-oriented systems. Issues of granularity of classification in the relational model led to decomposition of multilevel relations, and then to the concerns of polyinstantiation -- which some would say currently includes several distinguishable causes, but which have similar operational symptoms.

While the object model is more complex than the relational model, the use of object identity provides some control for polyinstantiation. The object model also highlights the fact that there is interdependence among the classification levels assigned to schema components and to object and attribute instances. Perhaps these classification levels of the model should be subject to *security constraints* so as to support a consistent security policy, as has been proposed.

The theme of constraints arises even more directly as a consequence of application semantics. Such semantic constraints span multiple levels and thus may conflict with security-based separation of information. Distributed database systems demonstrate an analogous conflict between physical separation of information due to distribution versus similar logical interdependence of data for semantic consistency. For distributed systems, secure design will add another dimension of complexity. It will be interesting to see how the techniques developed for single site multilevel security – especially the replicated approach – may be extend ed to distributed security.

At the heart of such conflicts with application semantics is the theme of data *integrity and consistency*. For example, polyinstantiation often conflicts with application semantics regarding uniqueness of keys and consistency of single-valued attributes. The security field is ripe for a renewed inquiry into the issues of data integrity and faithful modeling of the application. Also, the rapidly growing distributed information Web is providing an opportunity for security to support both commercial and government applications in a new kind of information network. Perhaps an ultimate challenge for security is whether it can contribute to the *safety* of individual systems and the safety of composed or interconnected systems.

Policy modelling:

Chair: V Varadharajan

Hewlett-Packard Labs., UK

Providing Consistent Views in a Polyinstantiated Database

Laurence Cholvy Frédéric Cuppens
ONERA-CERT
2 Av. E. Belin
31055, Toulouse Cedex
France
email: {cholvy,cuppens}@tls-cs.cert.fr

Abstract

In a situation where data are polyinstantiated, one problem which arises is that the high users can observe both the high level data and the low level cover story, the high data being contradicted by the cover story. If the database provides the high users with all these data without explanation, the high users would be faced with an inconsistent situation. In this paper, we propose a formal mechanism that enables the global consistency of the database to be restored. This mechanism is based on the merging of the high level view of the database with the lower level views by assuming that, when two contradictory facts exist in the database, the higher sensitive fact is the most reliable one and the lower fact is a cover story. However, in the case of a partial ordering of the security levels, the use of the order defined on the security levels is not sufficient to restore the database consistency. In this case, we suggest to associate topics with data for representing some semantic links between data. Then, we use topics to parameterize the order of the security levels in order to define a finer grain of preference when the database consistency is restored.

Introduction

Since its introduction in the Seaview Model in 1987 [DLS+87], polyinstantiation has generated a great deal of controversy. Much has been written on this topic, and several panels have been organized. Two extreme positions can be identified with respect to polyinstantiation:

1. Polyinstantiation is an inevitable phenomenon of multilevel data. It is a property of information and not of any specific technology. In this case, large numbers of polyinstantiated tuples are usually generated and the problem is to investigate how best to deal with these spurious tuples.
2. Polyinstantiation and integrity are fundamentally incompatible. The results of polyinstantiation are unacceptable for an operational system because it could prevent a job from being done properly. In this case, solutions must be found to avoid polyinstantiation.

Our view lies between these two extreme points. It is argued in [Bur90, Bur91] that cover stories are the only good reason for the use of polyinstantiation. We agree with the point of view that representing cover stories is an appropriate use of and motivation for polyinstantiation. However, it is important to understand that there is nothing fundamental about the occurrence of polyinstantiation. Jajodia and Sandhu [JS91, SJ91] have shown how it is possible to prevent polyinstantiation in many situations where there is no need for cover stories. Hence, polyinstantiation should only be used where it is appropriate.

When this point is made clear, several problems have to be solved. In a situation where data are polyinstantiated, the high users try to lie to the low users in order to cause them to believe something which is incorrect. The problem of how to properly choose a cover story is discussed in [GL92]. In particular, a cover story to be effective usually requires consistency. The second problem is that the high users are authorized to have a complete view of the database. They can observe both the high level data and the low level cover story, the high data being contradicted by the cover story. If the database provides the high users with all these data without explanation, the high users would be faced with an inconsistent situation.

In this paper, we propose a mechanism that allows us to restore the global consistency of the database. This mechanism is based on the merging of the high level view of the database with the lower level views by assuming that, in case of polyinstantiated data, the higher sensitive datum is the most reliable one and the lower datum is a cover story[1]. However, in specific situations, the use of the order defined on the security levels is not sufficient to restore the database consistency. These situations appear in the case of a partial ordering of the security levels. For these situations, we propose to associate some topics with data. Topics allow representation of some semantic links between data [CD88, CD89]. For instance, we can associate the same topic *Localization* with the two relations *Departure_City* and *Arrival_City*. We will show that in many cases it is actually possible to identify a topic with a set of information a user needs to know to perform a particular job. Then, we use topics to parameterize the order of the security levels. This allows us to define a finer order of preference when database consistency is restored.

The remainder of this paper is organized as follows. Section 1 reviews the concept of polyinstantiation emphasizing those aspects which are important to the objective of our paper. In section 2 we show, through examples, how to restore database consistency when polyinstantiation is used. Section 3 proposes a formal mechanism for restoring the database consistency. In section 4, we illustrate how to use this mechanism to provide answers to variously classified users. In section 5, we compare our approach with related work and section 6 concludes the paper on further work that remains to be done.

1 Polyinstantiation and cover stories

A common use for cover stories is to hide the existence of an otherwise sensitive event. For example, the plane F127 might be said to carry food when its actual cargo is gas-masks. Without a cover story, an unclassified user who asks the query *"What is the cargo carried by F127"* will be provided with the answer *"I don't know"* or *"You are not cleared to know that"*. In both cases, the fact that an answer is not provided may disclose the existence of the secret mission of F127. In many situations, this disclosure may not be desirable if a mission is to be successful.

However, notice that, in many other situations, there is no need for cover stories. For instance, let us consider a database containing information about the members of a Secret Service and let us assume that an unclassified user asks the query *"Give me the list of spies"*. As it is well-known by everybody that information related to a Secret Service is sensitive, we argue that the best answer to this question is *"You are not cleared to know that"*. Hence, a cover story should only be used where appropriate. This point has been noticed several times before (see for instance [Wis90, SJ92]).

On the other hand, polyinstantiation is a technique introduced by Denning et al. in [DLS+87].

[1] It is important to notice that we only make this assumption to compare two contradictory facts. It would generally be false to consider that the high level data are more reliable than the lower ones; confidentiality levels are not used to represent the data reliability.

It was used as a technique for closing a signaling channel which arises when an unclassified user inserts a tuple that has the same primary key values as an existing but higher sensitive tuple. This initial view of polyinstantiation has been discussed and many researchers ([Bur90, SJ92] for instance) consider that these technical arguments are not the best motivation for polyinstantiation. We agree with this point of view and, as [Bur90, SJ92], we consider that:

Claim 1 *Polyinstantiation is actually a technique that must only be used to support cover stories.*

However, Wiseman argued in [Wis92] that polyinstantiation is not essential for supporting cover stories and he showed that SWORD is perfectly capable of supporting cover stories without using polyinstantiation. Wiseman actually considers that polyinstantiation is a poor technique for cover stories because it is difficult to prevent them arising spuriously. His conclusion would be that polyinstantiation is a threat for the global integrity of the database. For instance, let us consider the scenario adapted from the one he proposed in [Wis90].

Example 1 *"Suppose an officer wishes to send gas-masks to the forces at The Front. He queries the database and discovers that aircraft F127 is suitable and available. He then "books" that aircraft by recording in the database that F127 is carrying gas-masks to the front. He decides that, for strategic reasons, this fact is Secret (..).*

Now suppose the system is also used by the Army Catering Corps to arrange delivery of rations to the troops. This activity is less sensitive than supplying armaments, so the officer in charge is only cleared to Confidential. Wishing to restock forces at headquarters with champagne, the catering officer queries the database and finds that aircraft F127 is suitable and available. He is not told that it is already booked because he is only cleared to Confidential and hence, because of polyinstantiation his query does not see the secret fact. Therefore the officer goes ahead and arranges for F127 to carry champagne to HQ. The database now contains two conflicting facts[2]*. (..) The database is therefore inconsistent."* □

If such a scenario could arise, then we would really be faced with an integrity problem because it is not clear to answer to the question *"Who will win?" the armament officer who wants to carry gas-masks to the front or the catering officer who wants to carry champagne to HQ*. However, it is not difficult to prevent this situation from occurring. When the armament officer inserts the secret fact that F127 is carrying gas-masks to the front, then there are two possibilities:

1. The armament officer wants to hide the existence of a secret mission for F127. Then, this officer himself must create a confidential session to insert a cover story to protect the existence of the secret information.
2. The armament officer does not want to hide the existence of a secret mission for F127. In this case, this officer must also create a confidential session to insert that F127 is booked for a secret mission. For this purpose we can use, as suggested in [SJ91], the special symbol *restricted* whose meaning is that some data exists but is higher classified.

In both situations, the catering officer will know that F127 is already booked. Deciding whether the catering officer is authorized to modify the mission arranged by the armament officer does not depend on the confidentiality policy, but depends on an integrity policy which defines who is permitted/prohibited to perform updates in the database. For instance, let us assume that the unclassified data (actually a cover story) *"F127 is carrying champagne"* is stored in the database and user A wants to update this data. Then, there are two possibilities:

1. The integrity policy says that A is prohibited to modify the cargo of F127. In this case, A's

[2] We actually assume that F127 can carry only one cargo and has only one destination.

update would be rejected because F127 is already booked. Notice that this does not represent a signaling channel; A's update is rejected because of the existence of an unclassified data.

- A is permitted to modify the cargo of F127. Let us assume that this update will represent an effective change of the real cargo. In this case, the unclassified cargo will be updated but it is likely to also update the secret cargo. No one has proposed this solution seriously because deleting the existing secret cargo would clearly represent a threat. However, by using an integrity policy, we argue that in many cases this solution becomes realistic: we have only to properly define who is permitted to perform the update.

Similarly, Sandhu and Jajodia introduced in [SJ91] special integrity privileges for changing restricted to unrestricted. However, it is not the purpose of this paper to further discuss integrity policies. We only want to justify our second claim:

Claim 2 *In case of polyinstantiated data, there must be only one person who controls the secret information and the associated cover story.*

This claim allows us to conclude that, when two contradictory facts exist in a database, the higher sensitive fact is always the most reliable one and the lower fact is a cover story.

Continuing the above example, Wiseman then wants to show that polyinstantiation is unacceptable because it might prevent a job from being done properly.

Example 1 (continued) *"Suppose that other officers use the database to receive their orders. The flight crew of F127 are cleared to confidential because they do not need to know about the cargo they are transporting. Therefore the database tells them they are going to HQ because the fact that they should be going to the Front is secret. Note that the crew are about to make a big mistake."* □

A possible solution to prevent this kind of mismatch is to introduce a set of compartments of information to create a finer grain of classification on the basis of *need-to-know*. For instance, in the above example, we can create two compartments *Destination* and *Freight*. The flight crew would be actually cleared up to $(Secret, Destination)$ and the data *"The destination of F127 is The Front"* would be also classified $(Secret, Destination)$. In this case, the flight crew would be told the proper destination of F127. On the other hand, the data *"The cargo of F127 is gas-masks"* would be classified $(Secret, Freight)$ and the flight crew would not know about the cargo they are transporting. Hence, we can state our third claim:

Claim 3 *When polyinstantiation is used, we must always consider the specific job-related need to know of users, i.e. do not provide a cover story to users if this lie would prevent them from properly performing their job.*

if this third claim is respected, then we argue that there is no fundamental incompatibility between polyinstantiation and integrity. Moreover, according to this claim, if two contradictory facts exist in a database and if a user needs to know one of those facts to properly perform his job, then the job related fact is the most reliable one and the other fact is a cover story.

2 Merging polyinstantiated databases

The main idea of this paper is to apply techniques of multi-sources reasoning [DDP92], [Cho92], [Cho94b], [Cho93] in order to provide a consistent view of the database when polyinstantiation is used.

2.1 Multi-source reasoning: state of the art

Roughly speaking, multi-source reasoning is a kind of reasoning with information provided by different sources of information. This can be seen as a way of merging several sources of information or databases. One problem which arises is that of inconsistency: even if each source is self consistent, the global set of information may be contradictory. One way of solving this problem is to consider that the sources are ordered according to a total order which expresses their relative reliability. [Cho94a] and [CD94] show that it is actually more judicious to consider as many orders as *topics*. Intuitively a topic is a cluster of formulas which "concern" the same thing. In this section, we will show that it is generally necessary to consider topic-dependent orders on the different databases. Furthermore, the previous works defined two attitudes called "suspicious" and "trusting". They differ in that when a source contradicts a more reliable source, we can either globally reject it or we can reject only the contradictory information. If topics are not used, then the trusting attitude is clearly more adequate because the suspicious attitude would cause all lower classified data to be ignored as soon as a contradiction appears between the lower classified database and the higher classified database. However, in using topics, we can define a finer order of preference when database consistency is restored. We will show in the following that when applying multi-source reasoning with topics to polyinstantiated database, the "suspicious" attitude becomes more adequate.

2.2 Application of the multi-source reasoning techniques to polyinstanciated databases: examples

We suggest applying the multi-source reasoning technique to restore database consistency when polyinstantiation is used. In a multilevel context, each piece of information is assigned a security level. Hence, one can partition the global multilevel database into single-level databases[3] corresponding to each security level. The data stored in each single-level database generally correspond to a partial view of the universe by users at the corresponding security level. Indeed, a user at a given security level is cleared to observe all the single-level databases which are dominated by the user's clearance. To provide the user with a complete view of the universe corresponding to his clearance level, we suggest merging all these single-level databases.

In the case of polyinstantiation, the multilevel database contains some contradictory facts. Our goal is to restore a consistent view of the universe at each security level. To achieve this goal, we make the following assumption:

Claim 4 *Each single-level database is internally consistent.*

Hence, if the multilevel database contains two contradictory facts, then these facts would belong to two different single-level databases. To achieve claim 4, we must precisely constrain the use of polyinstantiation (see [CY92] for a detailed discussion).

If claim 4 is respected, then we propose merging a given single-level database with all the lower single-level databases. If the security levels are totally ordered, then this principle allows us to provide a consistent view of the universe at each security level. It is illustrated in the following example:

Example 1 (continued) Let us consider the state of the database after the armament officer has stored in the database the secret fact that F127 is carrying gas-masks to the front. We also assume that the armament officer has inserted, in order to hide the secret mission for F127, the unclassified fact that F127 is carrying champagne to HQ. Finally, Captain Brown has been ordered to be the pilot

[3]The single-level databases are perhaps not real databases but only virtual databases

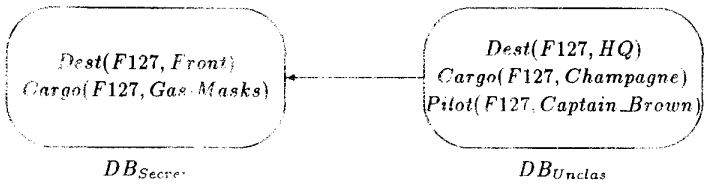


Figure 1: Complete view of the multilevel database

of F127. This last fact is also unclassified. These facts are respectively inserted into two single-level databases denoted DB_{Secret} and DB_{Unclas} (see figure 1).

Since there is no lower level than unclassified, the complete view of the universe by users at the unclassified level is equal to the single-level database DB_{Unclas}. But, to provide a secret user with a complete view of the universe, we need to merge DB_{Secret} with DB_{Unclas}. In merging $Dest(F127, Front)$ with $Dest(F127, HQ)$, a contradiction appears. However, as $Dest(F127, Front)$ is more sensitive than $Dest(F127, HQ)$, claim 2 allows us to conclude that $Dest(F127, Front)$ is more reliable than $Dest(F127, HQ)$. Hence, a secret user will be provided with $Dest(F127, Front)$. Similarly, after merging the two single-level databases, we will obtain $Cargo(F127, Gas\text{-}Masks)$ in the secret view. Finally, there is no fact in DB_{Secret} which contradicts that $Pilot(F127, Captain\ Brown)$. In this case, a secret user could adopt two different attitudes:

1. A suspicious attitude. As DB_{Unclas} contains facts which are contradicted by DB_{Secret}, the secret user does not believe in any facts stored in DB_{Unclas}.
2. A trusting attitude. The secret user believes in any facts of DB_{Unclas} which are not contradicted by DB_{Secret}.

In the sequel, we will discuss which attitude is the best one in the case of polyinstantiation. In our example, if a secret user adopts a trusting (resp. suspicious) attitude then he will (resp. not) believe that $Pilot(F127, Captain_Brown)$. Figure 2 shows the complete view of the universe by unclassified and secret users, we respectively denote them $DB_{\leq Unclas}$ and $DB_{\leq Secret}$, when the secret users adopt a trusting attitude. In this example, it seems that this attitude is more adequate because it allows the secret user to observe the likely correct information $Pilot(F127, Captain_Brown)$. □

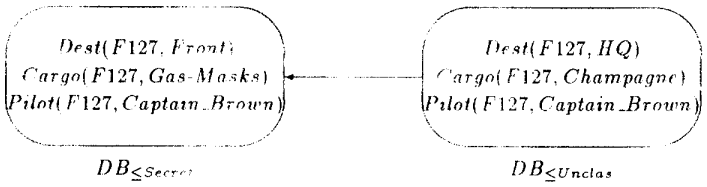


Figure 2: View at the unclassified and secret levels (trusting attitude)

However, a reader in search of simplicity may found that this internal merging of databases is useless and introduces unnecessary complications. He may consider, as most multilevel databases suggest, that it is sufficient to provide a secret user, who queries the multilevel database to know the destination of F127, with the two facts

$$Dest(F127, HQ), Dest(F127, Front)$$

and to mention that the first fact is unclassified and the second one is secret. Then, he will assume that this secret user can perform an "external" merging of the two facts and derive that $Dest(F127, Front)$ is the real fact and $Dest(F127, HQ)$ is a cover story. Unfortunately, this "solution" does not apply to many situations, especially in the case of deductive database. The following example illustrates this problem.

Example 2 Let us consider the following state of a multilevel database used in a travel agency:

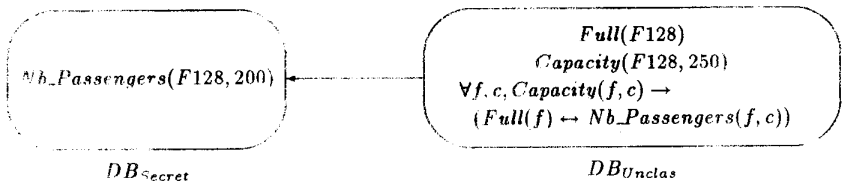


Figure 3: Complete view of the multilevel database

We assume that, in this travel agency, some seats are kept free for secret users. This is the reason why, the database stores the unclassified fact that F128 is full even though some seats are still available for secret users. If a secret user queries the database to know if F128 is full, then the database management system (DBMS) will answer *yes* and mention that this fact is unclassified. Notice that if we do not merge the secret database with some unclassified facts, then the DBMS cannot tell this secret user that F128 is actually not full and mention that this fact is secret. Hence, the DBMS only provides the secret user with the unclassified cover story! In using the approach we suggest in this paper, the complete view of the universe at the secret level is represented in the following database.

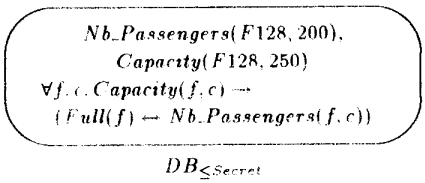


Figure 4: Complete view at the secret level

Now, if a secret user queries the database to know if F128 is full, then the DBMS also derive the answer *no* and mention that this fact is secret. □

Unfortunately, in case of a partial ordering of the security levels, it is not sufficient to use the order defined on the security levels to restore the database consistency. The following example illustrates this problem.

Exemple 1 (Continued) Let us assume that the flight crew of F127 use the database to receive their orders. To allow the flight crew to know about the destination but not about the cargo, we have created a compartment *Destination*. The flight crew are cleared up to $(Secret, Destination)$ and the data $Dest(F127, Front)$ is actually classified at $(Secret, Destination)$ to allow the flight crew to know about the proper destination of F127.

Similarly, let us assume that the ground crew use the database to know what cargo to load. We assume that they do not need to know the destination. Hence, we create another compart-

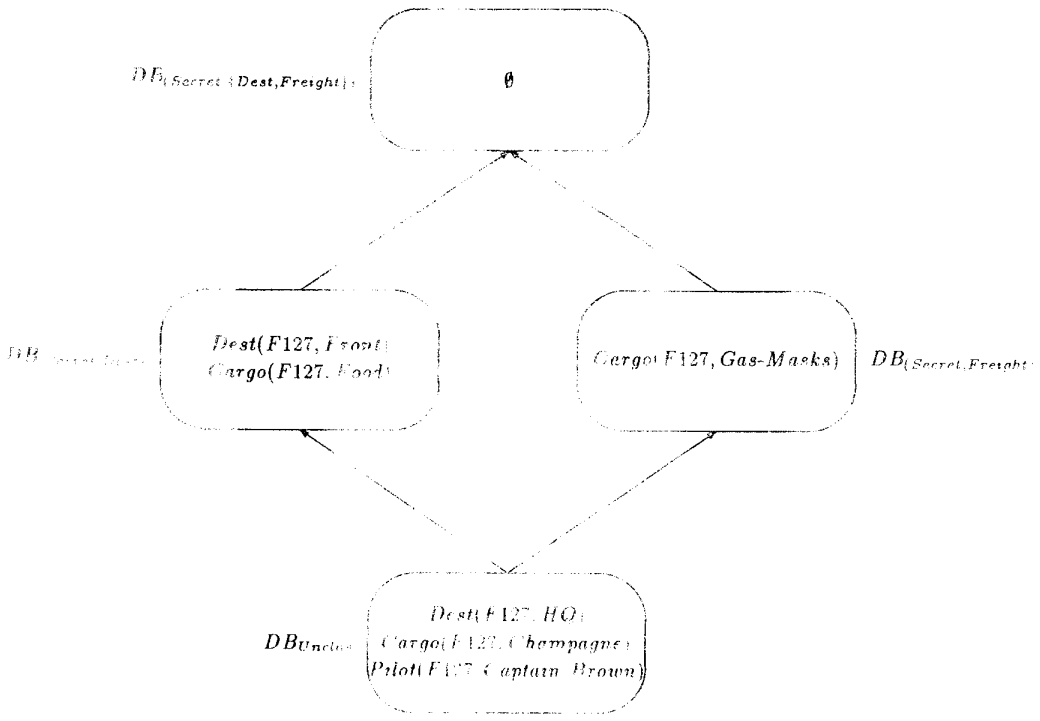


Figure 5: Complete view of the multilevel database

ment *Freight* the ground crew are cleared up to $(Secret, Freight)$ and the data $Cargo(F127, Gas\text{-}Masks)$ is classified at $(Secret, Freight)$.

We also assume that the following unclassified integrity constraint is stored in the database:

$$\forall x, Cargo(x, Champagne) \Rightarrow Dest(x, HQ)$$

This constraint says that champagne may only be sent to HQ. Let us now consider the flight crew's view of the universe. They can observe the exact destination of F127 – $Dest(F127, Front)$ – and the unclassified cargo of F127 – $Cargo(F127, Champagne)$. However, this view is contradicted by the above integrity constraint. By using claim 2, the flight crew know that $Dest(F127, Front)$ is the proper destination of F127. So, the flight crew will derive that $Cargo(F127, Champagne)$ is a cover story. A possible solution to prevent this kind of disclosure is to introduce a second cover story for the flight crew, for instance $Cargo(F127, Food)$. On the other hand, the ground crew can observe that $Dest(F127, HQ) \wedge Cargo(F127, Gas\text{-}Masks)$. In this case, we do not have to change the cover story $Dest(F127, HQ)$ because we consider that it is possible to restock the headquarter with gas-masks. Figure 5 sums up the complete view of the multilevel database.

In using a similar approach as the one used in the previous example, it is easy to build the complete view of the universe by users at the $(Secret, Destination)$ and $(Secret, Freight)$ levels. But, to provide a user cleared up to $(Secret, \{Dest, Freight\})$ with a complete view of the universe, we need to merge $DB_{(Secret,Dest)}$ with $DB_{(Secret,Freight)}$. In merging $Cargo(F127, Food)$ with

$Cargo(F127, Gas\text{-}Masks)$, a contradiction appears. As we cannot compare $(Secret, Destination)$ and $(Secret, Freight)$, a user cleared at $(Secret, \{Dest, Freight\})$ cannot use the order of the security levels to restore the database consistency. However, a user cleared at $(Secret, \{Dest, Freight\})$ knows that users cleared at $(Secret, Freight)$ need to know the correct cargo to properly perform their job. Hence, a user cleared at $(Secret, \{Dest, Freight\})$ can use this knowledge to deduce that $Cargo(F127, Gas\text{-}Masks)$ is the most reliable fact and, therefore, $Cargo(F127, Food)$ is a cover story.

We suggest introducing the concept of *topic* [CD89] to model the deduction the user at $(Secret, \{Dest, Freight\})$ has performed to derive $Cargo(F127, Gas\text{-}Masks)$. Data with similar semantics are linked by the same topic. In our example, we may introduce three topics: *Dest*, *Freight* and *Crew*[4]. We respectively associate the topic *Dest* with the data $Dest(F127, Front)$ and $Dest(F127, HQ)$, the topic *Freight* with $Cargo(F127, Gas\text{-}Masks)$, $Cargo(F127, Food)$ and $Cargo(F127, Champagne)$, and the topic *Crew* with $Pilot(F127, Captain_Brown)$.

In order to provide secret users with a consistent view of the database, we use topics to parameterize the order of the security levels and to merge data with this finer grain of preference. For instance, we define the following total order of preference[5] for merging information related to the topic *Dest*:

$$(Secret, \{Dest, Freight\}) >_{Dest} (Secret, Dest) >_{Dest} (Secret, Freight) >_{Dest} Unclas$$

In particular, this order means that information related to the topic *Dest* are more reliable at level $(Secret, Dest)$ than at $(Secret, Freight)$. Similarly, we define the following total order of preference for the topic *Freight*:

$$(Secret, \{Dest, Freight\}) >_{Freight} (Secret, Freight) >_{Freight} (Secret, Dest) >_{Freight} Unclas$$

Notice that this order differs from the one defined for the topic *Dest* because, according to the specific need to know of users at level $(Secret, Freight)$, information related to topic *Freight* are more reliable at level $(Secret, Freight)$ than at $(Secret, Dest)$. Finally, for the topic *Crew*:

$$(Secret, \{Dest, Freight\}) >_{Crew} (Secret, Dest) >_{Crew} (Secret, Freight) >_{Crew} Unclas$$

These orders of preference are used to restore database consistency at the $(Secret, \{Dest, Freight\})$ level. Figure 6 (next page) shows the resulting view of the database at the different security levels.

We have also pointed out that a user at $(Secret, \{Dest, Freight\})$ can adopt two different attitudes: a suspicious attitude or a trusting attitude. Which attitude is the best in case of polyinstantiation? If we do not use topics, then we have already noticed that the trusting attitude is probably more adequate. However, we think that when using topics, the best attitude is the suspicious one. Indeed, let us consider the following example: at the unclassified level, we insert that the champagne carried by F127 is a *Veuve Cliquot of 1981*. Clearly, we do not want to provide the secret user with the fact that F127 is actually carrying gas-masks called *Veuve Cliquot* and dated from 1981. So, if a contradiction appears between the lower information related to a given topic and the higher information related to the same topic, then the best attitude is to reject all the lower information related to this topic because these information are probably related to the same cover story - namely that F127 is carrying champagne in our example. This attitude does not preclude the high user from observing lower information related to another topic, for instance *Crew*, and thus knowing that the Pilot of F127 is Captain Brown. Hence, in separately merging information related to the same topic, we argue that the best attitude is the suspicious one.

[4] The sets of compartments and topics may generally overlap.

[5] Actually, this order would be defined by the database security administrator.

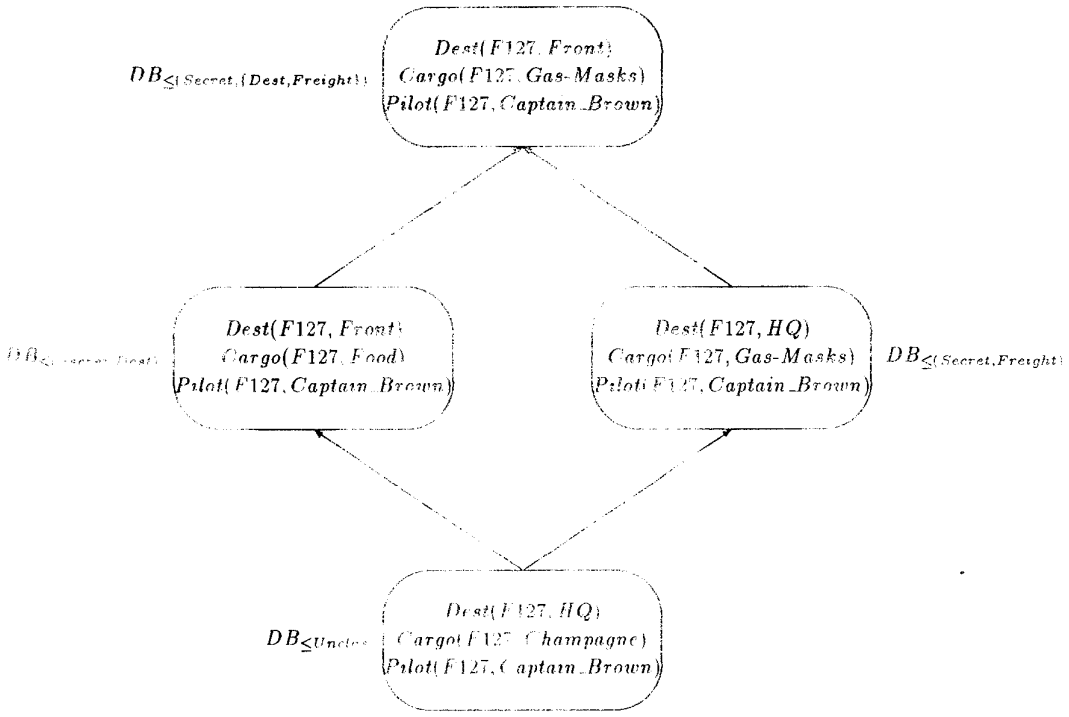


Figure 6. View at the different security levels

Moreover, for the readers who might prefer the trusting attitude, it is important to notice that the database administrator who is in charge of defining topics may choose a very general or a more specific representation. In particular, if he decides to associates any pair of literals $(l, \neg l)$ with a a topic then the suspicious attitude and the trusting attitude collapse. Hence, the trusting attitude may be seen as a particular case of the suspicious attitude □

In the next section, we propose a formal model for restoring the multilevel database consistency which includes topics and adopts a suspicious attitude

3 The formal model

3.1 Assumptions of our model

- **Security levels**

  Let us first assume the existence of a set of security levels which are ordered according to a strict partial order, noted $>$

  $level_1 > level_2$ means that people cleared up to $level_1$ may access information which belong to $level_1$ or to $level_2$. People cleared up to $level_2$ are not allowed to access information at $level_1$.

  For instance, the four levels: $secret_{1,2}$, $secret_1$, $secret_2$, $unclas$ may be partially ordered in the

following way: $secret_{1,2} > secret_1 > unclas$, and $secret_{1,2} > secret_2 > unclas$.

- **The language and the approach.**
  We assume that the language used to describe the databases is propositional. Notice that, even if apparently a first order language is needed (tuples of relations) we can associate it with a propositional one because of the domain closure (i.e. we assume that there is a finite number of objects)[Rei78].
  The databases which we consider here are sets of propositional literals and we adopt a model-theoretic approach, i.e. each database is associated with its logical models. Notice that this restriction to literals means we cannot represent rules in the database, such as those in example 2. This clearly represents some further work that remains to be done.

- **Topics**
  We assume that the underlying propositional language is partitioned in several pairwise disjoint subsets called "topics". In this paper, we do not consider the case of topics structured with an ISA relation: we only consider that topics form a partition of the language. The only condition we impose on topics is the following one. Let t be a topic, let f be a formula:

  $$(f \in t) \Longleftrightarrow (\neg f \in t)$$

  This condition says that the formula f belongs to the topic t if and only if the negated formula $\neg f$ belongs to t.
  Each topic is supposed to be a set of formulas which "concern" the same thing. For instance the following two formulas: "the destination of F127 is the front" and "the destination of F127 is the head-quarter" belong to the same topic: "destination-of-F127". But "the pilot of F127 is Captain Brown" does not belong to this topic. Notice however that the definition of topics depends on the context. Indeed, if it was necessary, we could have considered only one topic "F-127" which could have grouped the three previous formulas.

- **Topic-dependent orders.**
  As said previously, each security level is associated with a (possibly virtual) database. For instance, in the above example, there would be four databases, respectively accessed by the people cleared up to $secret_{1,2}$, $secret_1$, $secret_2$, and $unclas$.
  The existence of cover-stories make these databases apparently inconsistent. The solution we suggest is to allow the database security administrator to express topic-dependent orders on security levels.
  Let t be a topic, we note $>_t$ a total order of levels which is associated with t.

  Definition 1 *Compatibility between security levels and topic-dependent orders.*
  Let $>$ be the partial order defined on the security levels. Let $>_t$ be a topic-dependent total order. They are compatible iff for all levels l_1 and l_2 we have: $(l_1 > l_2) \Longrightarrow (l_1 >_t l_2)$.

  In other words, $>$ and $>_t$ are compatible iff $>_t$ is a total extension of $>$.

  Claim 5 We assume that the different topic-dependent orders $>_t$ are compatible with $>$.

3.2 Semantics of the suspicious fusion with topic-dependent orders

Consider n databases to be combined. Let us note L the underlying propositional language associated with m topics $t_1, ..., t_m$. The individual databases are finite sets of literals of L which are satisfiable (consistent) but not necessarily complete (a base B is not complete if there is at least one literal l such that $l \notin B$ and $\neg l \notin B$).

In this section, we give the semantics of a logic, called FUSION-S whose language L' is obtained from L by adding pseudo-modalities i.e., marks on propositional formulas. These pseudo-modalities are:

- $[O_1 \ldots O_m]$, where the O_i are total orders on a given subset of $\{1, ..., n\}$ which are t_i-dependent and compatible with $>$.

Let $O_1 \ldots O_m$ be m total orders on k databases. Our goal in this section is to define a semantics for $[O_1 \ldots O_m]F$. Intuitively, $[O_1 \ldots O_m]F$ means that F is true in the database obtained by virtually merging the k databases according to the m t_i-dependent orders $O_1 \ldots O_m$. The satisfiability of $[O_1 \ldots O_m]F$ is defined in definition 5.

Remark Notice that the general form of these pseudo-modalities allows us to represent the particular case where $k = 1$. In this case, there is only one database i_1 to be ordered.
So, $O_1 = O_2 = \ldots = O_m = (i_1)$ And $[i_1 \ldots i_1]F$ will mean that F is true in database i_1.
By convention, $[i_1 \ldots i_1]$ will be noted $[i_1]$

Definition 2 Let m be an interpretation[6] of L and t a topic of L. We define:

$$m \mid t = \{l : l \in m \text{ and } l \in t\}$$

$m \mid t$ is the projection of the content of m on the topic t

Definition 3 Let E be a set of interpretations of L and t a topic of L. We define:

$$E \mid t = \{m \mid t : m \in E\}$$

Definition 4 Let t be a topic and O_t be a total order $(i_1 > \ldots > i_k)$ on k databases. We define :

$R_t(O_t) = h_{i_k,t}(\ldots h_{i_2,t}(R(i_1))\ldots)$, where: $h_{i_j,t}(E) = R(i_j) \mid t \cap E \mid t$ if not empty
$h_{i_j,t}(E) = E \mid t$ else

Definition 5 The unique model of FUSION is the pair $M = (W, \mathcal{R})$ where:

- W is the finite set of all the interpretations of L.
- R is a finite set of subsets of W such that any modality $[O_1 \ldots O_m]$ is associated with such a subset noted $R(O_1 \ldots O_m)$. These subsets are defined by :

  $R(i_1 \ldots i_1)$ is the set of models of dabatase i_1. We note it $R(i_1)$.
  $R(O_1 \ldots O_m) = \{w : w = w_1 \cup \ldots \cup w_m$, where
  $\forall i \in [1..m], w_i \in R_i(O_i)$ and
  $\forall l \in L, l \notin w$ or $\neg l \notin w\}$

We can prove that

Proposition 1 $R(O_1 \ldots O_m)$ is never empty

This means that, although the bases are contradictory (because of the cover stories), the combined base is not contradictory (i.e the information provided at any level will not be contradictory).

[6] We assimilate an interpretation of L with a set of literals.

Definition 6 *(Satisfaction of formulas).*
Let F be a formula of L. Let F_1 and F_2 be formulas of L'. Let $O_1...O_m$ be total t_i-dependent orders on a subset of $\{1, ..., n\}$. Let $M = (W, \mathcal{R})$ be the unique model of FUSION and let $w \in W$.

$$\begin{array}{lll} \text{FUSION},w \models F & \Longleftrightarrow & w \models F \\ \text{FUSION},w \models [O_1...O_m]F & \Longleftrightarrow & \forall w', w' \in R(O_1...O_m) \Longrightarrow w' \models F \\ \text{FUSION},w \models \neg F & \Longleftrightarrow & (\text{FUSION}, w \not\models F) \\ \text{FUSION},w \models F_1 \wedge F_2 & \Longleftrightarrow & (\text{FUSION},w \models F_1) \text{ and } (\text{FUSION},w \models F_2) \end{array}$$

We note $M \models F$, iff $\forall w \in W$, FUSION, $w \models F$.

We are interested in finding formulas of the form: $[O_1...O_m]F$ which are satisfied in the model M, i.e., finding formulas F which are true in the database obtained by merging the databases, when the order is $O_1...O_m$

3.3 Model application

Example Let us consider again the previous example. Let L be the propositional language whose propositions are: front, HQ, food, gas-masks, champagne, captainbrown. Let us define three topics: Dest = {front, HQ}. Freight = {food, gas-masks, champagne} and Crew = {captainbrown}. We consider the three topic-dependent orders : $>_{Dest}, >_{Freight}, >_{Crew}$ defined by :

O_{Dest} : $(Secret, \{Dest, Freight\}) >_{Dest} (Secret, Dest) >_{Dest} (Secret, Freight) >_{Dest} Unclass$,

$O_{Freight}$:
$(Secret, \{Dest, Freight\}) >_{Freight} (Secret, Freight) >_{Freight} (Secret, Dest) >_{Freight} Unclass$,

O_{Crew} : $(Secret, \{Dest, Freight\}) >_{Crew} (Secret, Dest) >_{Crew} (Secret, Freight) >_{Crew} Unclass$,

Let us compute $R(O_{Dest}, O_{Freight}, O_{Crew})$ in the model M.

- $R(DB_{(Secret,\{Dest,Freight\})})$ is all the models of L which satisfy the integrity constraints expressing that a plane can carry only one freight and has only one destination.
- $R(DB_{(Secret,Dest)})$ = { (front, food, ¬ HQ, ¬ gas-masks, ¬ champagne, captainbrown),
  (front, food, ¬ HQ, ¬ gas-masks, ¬ champagne, ¬ captainbrown) }
- $R(DB_{(Secret,Freight)})$ = { (front, gas-masks, ¬ HQ, ¬ food, ¬ champagne, captainbrown }
  (HQ, gas-masks, ¬ front, ¬ food, ¬ champagne, captainbrown)
  (front, gas-masks, ¬ HQ, ¬ food, ¬ champagne, ¬ captainbrown)
  (HQ, gas-masks, ¬ front, ¬ food, ¬ champagne, ¬ captainbrown) }
- $R(DB_{Unclass})$ = { (HQ, ¬ front, champagne, ¬ gas-masks, ¬ food, captainbrown) }

Let us now compute $R_{Dest}(O_{Dest})$, $R_{Freight}(O_{Freight})$, $R_{Crew}(O_{Crew})$.

- $R_{Dest}(O_{Dest})$ = { front, ¬ HQ }
- $R_{Freight}(O_{Freight})$ = { gas-masks, ¬ food, ¬ champagne }
- $R_{Crew}(O_{Crew})$ = { captainbrown }

Thus, by definition 5 :

- $R(O_{Dest}, O_{Freight}, O_{Crew})$ = { front, ¬ HQ, gas-masks, ¬ food, ¬ champagne, captainbrown }.

So, $M \models [O_{Dest}O_{Freight}O_{Crew}]$ (front ∧ gas-masks ∧ captainbrown),

i.e. the formula (front ∧ gas-masks ∧ captainbrown) is true in the base obtained by merging all the bases which are under $DB_{(Secret,\{Dest,Freight\})}$. In other terms, a person who is cleared up to $(Secret, \{Dest, Freight\})$ will know that *"Captain Brown pilots the F-127 to the front with a cargo of gas-masks"*.

4 Answering queries

In this section, we show how to answer queries addressed to the database which is composed of several databases attached to different security levels. This query evaluator is based on the semantics given in the previous section.

First of all, we need to introduce the following definitions:

Definition 7 Let $DB_{l_1}...DB_{l_n}$ be n databases associated with n security levels. Let O_t be a total order on these databases which is dependent on a topic t. Let us note $O_t = \{ l_1 >_t l_2 >_t ... >_t l_n \}$. Let l be a security level. We define $O_t|l$ the restriction of O_t to the set of databases DB_{l_i} such that $l \geq l_i$.

Example Let us come back to the example introduced in section 3.3 and consider the security level $(Secret, Freight)$. For every topic $t \in \{Dest, Freight, Crew\}$, we have:

$$O_t|(Secret, Freight) = \{(Secret, Freight) >_t Unclass\}$$

Let us consider a person who is cleared up to level l asking a query Q (which is a formula of L). We define the answer of Q provided for persons cleared up to l as follows:

Definition 8 *(Answer to Q provided to persons cleared up to l)*
Let $DB_{l_1}...DB_{l_n}$ be n databases associated with n security levels. Let $O_{t_i}, i = 1, ..., m$, be total orders on these databases which are dependent on topic $t_1, ..., t_m$. Let Q be a formula of L.

$||Q||_l$ = TRUE iff $M \models [O_{t_1}|l...O_{t_m}|l]Q$
$||Q||_l$ = FALSE iff $M \models [O_{t_1}|l...O_{t_m}|l]\neg Q$
$||Q||_l$ = ? iff else

Example : The following array lists some questions and the answers provided according to the clearance level of the person who asks it. Notice that we give the queries in the first order language from which L is based

Habilitation	Queries		
	Where does F127 go?	What does F127 carry?	Who is the pilot of F127?
(Secret,{Dest,Freight})	Front	Gas-Masks	Captain Brown
(Secret,Dest)	Front	Food	Captain Brown
(Secret,Freight)	HQ	Gas-Masks	Captain Brown
Unclas	HQ	Champagne	Captain Brown

5 Comparison with related work

There exists some connection between our approach and the Nonmonotonic Typed Multilevel Logic (NTML) developed by Thuraisingham [Thu91]. In NTML, the primitive symbols such as constant, variable or predicate name are associated with security levels. Hence, it would be possible to consider that the fact $Cargo(F127, Gas\text{-}Masks)$ is secret because $Gas\text{-}Masks$ is a secret constant. In our approach, we consider that the logical language used to model the multilevel database is not protected. We agree that hiding some part of this language allows us to represent additional ways of protection. We consider that this problem represents further work that remains to be done. However, it is not essential to achieve our goals in this paper.

On the other hand, we feel that it is a conceptual simplification to represent a multilevel database as a set of single-level databases instead of a single multilevel theory as in NTML. We also prefer representing each single-level database as a set of models instead of as a theory. Moreover, in many specific situations it is not clear how to restore the database consistency using NTML. These situations include the case of a partial ordering of the security levels but are not restricted to this case. For instance, as noticed in [GLQS92], if P and Q are facts at the unclassified level and $\neg(P \wedge Q)$ a new fact at the secret level, it is not clear how to choose which of P and Q is not inherited from unclassified level to secret level to avoid a contradiction. In our approach, we propose to use topics to define a finer grain of preference which allows to restore the database consistency even though the security levels are partially ordered. Finally, information related to the same topic are separately merged. We argue that in this case the best attitude is a suspicious attitude instead of the trusting attitude used in NTML. Hence, in the above example, if P and Q are facts at the unclassified level and related to the same topic, $\neg(P \wedge Q)$ a new fact at the secret level, and the secret user adopts a suspicious attitude, then this secret user would reject both P and Q.

6 Conclusion

Situations exist where we need to hide the existence of an otherwise sensitive event. In these particular situations, we generally need to use cover stories. Hence, cover stories are a fundamental multilevel requirement. On the other hand, we agree with [Bur91] that polyinstantiation is not a fundamental property of multilevel databases; it is simply a powerful technique for supporting cover stories. Moreover, in using polyinstantiation, several problems have to be solved. This paper aims to solve one of these problems, namely how to restore the database consistency when polyinstantiation is used. We have proposed a formal mechanism which works even though the security levels are partially ordered. Further refinements of our approach are possible. A first refinement would be to extend our model to include the possibility to deal with rules in the database. This extension would allow us to treat the example 2 in our model.

We have also suggested that it could be interesting to hide some parts of the database schema in order to represent additional ways of protection. Another refinement would be to include the case where we do not need polyinstantitation, i.e. we do not want to hide the existence of a secret event. For this purpose, we can use the special symbol *restricted* introduced in [SJ91]. We do not feel that it would generate any problem because, in this case, the high view is not contradicted by the lower view. A third possibility would be to deal with content-dependent rules. For instance, we may introduce a rule saying that *"The destination of F127 is always secret information"*. All these refinements would allow to have a complete representation of the different ways of protection.

Our approach is designed to provide the high level user with some parts of the low level database which are not in contradiction with the high level database. It would also be interesting to extend

this approach so that it would be possible for the high level user to see what the lower users see and know if they are believing some cover stories. As our approach is designed to recognize if a given data is a cover story, we guess that it would probably be easy to include this extension in our model.

Another problem not discussed in this paper is how to properly choose a cover story. For instance, in figure 3, we have rejected $Dest(F127, Front) \wedge Cargo(F127, Champagne)$ because this view would be contradicted by an integrity constraint. Hence, a cover story to be effective requires consistency. On the other hand, we have accepted the view $Dest(F127, HQ) \wedge Cargo(F127, Gas\text{-}Masks)$ for users cleared up to $(Secret, Freight)$ because we have considered that this view may be plausible. Knowing if users at $(Secret, Freight)$ would really believe in this cover story is a difficult problem discussed in [GL92]. We feel that combining the solution to this last problem with the mechanism developed in this paper would be an important step toward a meaningful semantics for cover stories and therefore polyinstantiation.

Bibliographie

[Bur90] R. K. Burns. Integrity and secrecy. Fundamental conflicts in the database environment. In *Proceedings of the Third RADC Database Security Workshop*, 1990.

[Bur91] R. K. Burns. Polyinstantiation. A position statement. In *Proc. of the computer security foundations workshop*. Franconia, 1991. Panel position paper on Polvinstantiation.

[CD88] F. Cuppens and R. Demolombe. Cooperative Answering: a methodology to provide intelligent access to Databases. In *Second International Conference on Expert Database Systems*. Tysons Corner, Virginia, 1988.

[CD89] F. Cuppens and R. Demolombe. How to recognize topics to provide cooperative answering. *Information Systems*, 14(2), 1989.

[CD94] L. Cholvy and R. Demolombe. Reasoning with information sources ordered by topics. In *Proceedings of Artificial Intelligence : methodologies, systems and applications (AIMSA)*, 1994.

[Cho92] L. Cholvy. Consistency of merged databases. In *Proceedings of the Workshop on Cooperation systems*, Keele University (GB), 1992.

[Cho93] L. Cholvy. Proving theorems in a multi-sources environment. In *Proceedings of IJCAI*, 1993.

[Cho94a] L. Cholvy. Fusion de sources d'informations ordonnées en fonction des thèmes. In *congrès AFCET RFIA (Paris)*, 1994.

[Cho94b] L. Cholvy. A logical approach to multi sources reasoning. In *Lecture notes in Artificial Intelligence*, number 808. Springer-Verlag, 1994.

[CY92] F. Cuppens and K. Yazdanian. A "Natural" Decomposition of Multi-level Relations. In *IEEE Symposium on Security and Privacy*, Oakland, 1992.

[DDP92] J. Lang D. Dubois and H. Prade. Dealing with multi-source information in possibilistic logic. In *Proceedings of ECAI*, 1992.

[DLS+87] D. Denning, T. Lunt, R. Shell, M. Heckman, and W. Shockley. A Multilevel Relational Data Model. In *IEEE Symposium on Security and Privacy*, Oakland, 1987.

[GL92] T. D. Garvey and T. F. Lunt. Cover Stories for Database Security. In S. Jajodia and C. Landwehr, editors. *Database Security, 5: Status and Prospects*. North-Holland, 1992. Results of the IFIP WG 11.3 Workshop on Database Security.

[GLQS92] T. Garvey, T. Lunt, X. Qian, and M. Stickel. Toward a Tool to Detect and Eliminate Inference Problems in the Design of Multilevel Databases. In *Proc. of the Sixth IFIP WG 11.3 Working Conference on Database Security*, Vancouver, 1992.

[JS91] S. Jajodia and R. Sandhu. Enforcing Primary Key Requirements in Multilevel Relations. In *Proceedings of the Fourth RADC Database Security Workshop*, 1991.

[Rei78] R. Reiter. Deductive question-answering on relational database. In *Logic and data bases*. Plenum Press New-York, 1978.

[SJ91] R. Sandhu and S. Jajodia. Honest Databases That Can Keep Secrets. In *Proceedings of the 14th National Computer Security Conference*, Washington, D.C., 1991.

[SJ92] R. Sandhu and S. Jajodia. Polyinstantiation for cover stories. In *European symposium on research in computer security*, Toulouse, France, 1992. AFCET.

[Thu91] B. Thuraisingham. A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Database / Knowledge-Based Management Systems. In *Proc. of the computer security foundations workshop*. Franconia, 1991.

[Wis90] S. Wiseman. On the Problem of Security in Data Bases. In S. Spooner and C. Landwehr, editors, *Database Security, 3: Status and Prospects*. North-Holland, 1990. Results of the IFIP WG 11.3 Workshop on Database Security.

[Wis92] S. Wiseman. Using SWORD for the Military Aircraft Command Example Database. In *Proc. of the Sixth IFIP WG 11.3 Working Conference on Database Security*, Vancouver, 1992.

Secure Logic Databases Allowed to Reveal Indefinite Information on Secrets

Adrian Spalka

Department of Computer Science III, University of Bonn
Römerstr. 164, D-53117 Bonn, Germany
Fax: - 49 - 228 - 550 382, Email: adrian@cs.uni-bonn.de

Abstract

This paper presents a new approach to the definition and the enforcement of confidentiality in logic databases. We investigate the semantics of a logic database consequent upon the introduction of users and rights. Regarding a database with rights as a proper extension of an open database, we define the notion of global validity and that of a personal database profile. We give four formal definitions of confidentiality and show that three of them are meaningful in the presence of the Closed World Assumption. External assumptions regarding the allocation of rights and the individual user knowledge round up the formalism. Thereafter we focus our attention on the enforcement of the confidentiality-form G1, which allows a user to possess indefinite information on secrets. This form, being the lowest possible level of confidentiality, has the advantage that the database never lies to a user, ie it can be met without a cover story. We consider the enforcement of G1 with respect to the data and the integrity constraints. The presented formalism is theoretically sound, completely embodied in standard predicate logic and extendible to a multilevel security model.

1 Introduction

In this section we give an informal definition of an open logic database (LDB) and that of a secure LDB, viz a LDB which is expected to keep some confidential information

secret from particular users. Thereafter, we discuss previous works and related approaches. In this paper we put the emphasis on a clear illustration of our ideas, thus we omit some formal details.

1.1 Overview

A state of the world as seen by a LDB consists of facts, rules and general laws. The LDB-model maps the facts and rules of a state of the world into a set of data and the general laws into a set of static integrity constraints. A LDB uses normal clauses for the uniform representation of facts, rules, integrity constraints and queries. The application-dependent symbols which may occur in a clause are contained in the database's signature. The database-language comprises all clauses which the database understands, viz recognises. A user can directly obtain information from a LDB in two ways: he can request a listing of the current state's data and integrity constraints and submit a query which is evaluated under the Closed World Assumption* . The listing contains all clauses explicitly stored in the database; the answer to a query comprises facts either stored or derived from the data.

The life of a LDB is determined through a series of states. The application-dependent components which remain invariable with respect to all states, ie the signature and the integrity constraints, define a LDB-scheme. Two states of a LDB can only differ in their sets of data. A data set is consistent if it satisfies the integrity constraints, viz the data seen as a set of axioms must allow the derivation of the constraints. A state of a LDB is always valid, ie its set of data is consistent. Thus the set of all valid data sets is uniquely determined by the integrity constraints. A transaction is an activity which modifies the explicitly stored data of the present state. If the modified data set is valid, then the transaction is accepted and the LDB changes its state, ie the modified data set forms

* cf Reiter (1978).

the new LDB-state. Otherwise the transaction is rejected and the LDB-state remains unchanged, ie the modification is ignored.

In this open LDB, each user who has access to it can obtain a full listing of the stored data and integrity constraints, receive a complete answer to any query and change the LDB's state into any valid state. An open database treats all users equally or, to say it in another way, it can only discern one universal user with unrestricted power.

There are many reasons why it may be necessary to restrict the freedom of a user. One reason is the requirement to keep some elements of a LDB secret from a user. In the broadest sense we define a secure LDB as a LDB together with a set of users who have access to it and a set of confidentiality requirements that state which elements of the LDB should be kept secret from which user. The confidentiality requirements form a part of a security policy.

1.2 Related work

According to Gougen/Meseguer (1984), a confidentiality requirement expresses that "under certain conditions, certain individuals *should not* have access to certain information". Its formalisation as non-interference is specifically intended to model trusted processes, but the authors also introduce a simple model of a multilevel-secure database in proof-theoretical view which has neither integrity constraints nor updates, and where the Closed World Assumption (CWA) is not made. In this context, they interpret non-interference as non-derivability and say that a security violation has occurred if the data accessible at a low security level allow the derivation of data accessible only at a high security level. However, they do not mention until when a security violation has not occurred. Yet this distinction is important since $\alpha \in Th(I)$ and $\alpha \notin Th(I)$ are not the only relationship-possibilities between a formula and a set of theorems.

Gougen/Meseguer (1984):75

Berson/Lunt (1987a) and Berson/Lunt (1987b) investigate the possibility of the application of the MAC-model to deductive databases. They point out many new problems and suggest an approach to tackle them, but, due to the initial nature of these works, no solutions are offered.

Morgenstern (1987) notes that, in a deductive database, making a piece of information directly inaccessible may not be sufficient if one wants to keep it secret. Here it can still be possible to infer a secret from the accessible facts, rules and integrity constraints. The author speaks of deductive databases in an informal manner and uses them mainly to accentuate the new problems which arise during the transition from relational to deductive databases.

Meadows/Jajodia (1987), Burns (1990) and Wiseman (1990) are examples of early approaches which consider a multilevel relational database where primary key and foreign key constraints are the only classes of integrity constraints. They assume that each user has access to a different part of the database, but that there is just one set of constraints which must be satisfied at all levels. Burns (1990) and Wiseman (1991) note that there is a fundamental conflict between secrecy and integrity, since each of them can only be enforced at the expense of the other.

Lunt/Millen (1989), Garvey/Lunt (1990) and Garvey/Lunt (1991) choose an approach which considers deductive databases as a special case of object-oriented databases. Although their motivation has its origins in deductive databases, the presentation is based on the terminology of object-oriented databases. Hence it is difficult to regard this approach as a contribution to a predicate-logic based theory of secure databases.

Steinke (1991) reports on a specific problem in multilevel secure deductive databases which arises when integrity constraints must be kept secret. He argues that it may be impossible to keep a database valid. However, no satisfactory solution is offered.

Cuppens/Yazdanian (1991) extend a relational database with horn-clauses. Similar to Morgenstern (1987), the authors consider the 'inference problem', viz how to avoid that a user can draw a conclusion from his accessible data which should be kept secret from him. Rather than present a solution, they emphasise that logic is a suitable framework for the study of security problems in databases.

The first basic attempt of a formal treatment of confidentiality is presented in Thuraisingham (1991). The author's main idea is to formalise the multilevel security properties in NTML, a non-monotonic logic. Although this approach points to the right direction, NTML has been shown to be not sound.[1]

The advances achieved until 1991 are described in Tener (1991) as simply unsatisfactory. Security requirements have been so far neglected in the development of deductive databases, while at the same time deductive databases are increasingly employed in sensitive areas. Consequently, he urges a stronger formal security research.

The work of Bonatti/Kraus/Subrahmanian (1992) deals with the confidentiality of formulae in deductive databases. A formula is secret if it is not derivable. The authors allow the database to lie; their formalism and results are based on a mixture of standard predicate and an extended modal logic. However, the presented approach has some weak points. On the one hand, the authors define a very simple database model which lacks the CWA, integrity constraints and update operations. Yet, in our opinion, these are exactly the components which make the confidentiality-problem interesting. On the other hand, they make rather unrealistic assumptions on a user's own knowledge. These circumstances and some of their implications confine this work's applicability to a narrow context. Finally, no motivation is provided for the choices made in this approach, eg the unit of protection, the range of answers and the preference or necessity of modal logic in comparison to standard predicate logic.

---

[1] Garvey et al (1992):160.

Finally, the reliability of databases is discussed in Williams (1992). His arguments are based on the notion of external consistency.* It states that all data in a database visible to a user must be accurate. This requirement is as such always satisfied in a normal open database and to a large extent in a secure database allowed to reveal indefinite information on secrets to the user. However, it precludes the use of aliases which may be necessary in order to attain an even higher degree of confidentiality.

2 Basic definitions

Following Gallaire/Minker/Nicholas (1984) and Cremers/Griefahn/Hinze (1993) we consider databases from the viewpoint of predicate logic. Thus the discussion and the results are also valid for relational databases in proof-theoretical representation.†

2.1 Predicate logic

Definition 1 A signature Σ is a pair $\Sigma = (\mathbf{FS}, \mathbf{PS})$. The set **FS** contains ranked function symbols and **PS** ranked predicate symbols. Both sets, **FS** and **PS**, are non-empty, finite and disjunct.

Definition 2 The set of terms over the signature Σ, $\mathbf{TE}^{\Sigma}$, is the smallest set with the following properties: each variable is a term; each constant, ie a function symbol of rank 0, is a term; let f be a function symbol of rank k and $t_1, \ldots, t_k$ terms, then $f(t_1,\ldots,t_k)$ is a term. A term is ground if it does not contain any variable.

Definition 3 Let r be a predicate symbol of rank k and $t_1,\ldots,t_k$ terms, then $r(t_1,\ldots,t_k)$ is an atomic formula, or simply an atom. An atom is ground if it comprises only ground terms. Let α be an atomic formula, then α is also a positive literal and $\neg\alpha$ a negative literal. We denote the set of atomic formulae over Σ by $\mathbf{AF}^{\Sigma}$ and the set of literals over Σ

---

* cf Williams (1992):58.

† cf Reiter (1984).

by $\mathbf{LIT}^{\Sigma}$.

Definition 4 A clause is a formula of the form $\alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$, in which all variables are assumed to be universally quantified. Each α_i in the head of the clause is an atom and each β_j in its body a literal. A clause is ground if it comprises only ground atoms. A clause is normal if $m = 1$; it is a query if $m = 0$. A normal clause is called a rule if $n \geq 1$; it is called a fact if $n = 0$. A clause is range-restricted if each of its variables occurs also in a positive literal in its body. We denote the set of all range-restricted clauses over Σ by $\mathbf{CL}^{\Sigma}$ and its subset of normal clauses by $\mathbf{NCL}^{\Sigma}$.[*] ◉

We assume in this paper that all formulae are range-restricted clauses.

Definition 5 Let $X \subseteq \mathbf{CL}$ be a set of clauses, then $Th(X) \subseteq \mathbf{CL}$ denotes all clauses which can be (logically) derived from X (for a clause φ, $\varphi \in Th(X)$ is also denoted as $X \vdash \varphi$). The set of all literals in $Th(X)$ is denoted by $F(X)$, ie $F(X) = Th(X)\|_{\mathrm{LIT}}$.

2.2 Logic databases

Definition 6 A LDB-scheme DB is $DB = (\Sigma, C)$, where $\Sigma = (\mathbf{FS}, \mathbf{PS})$ is a signature and $C \subseteq \mathbf{CL}^{\Sigma}$ a set of static integrity constraints; the present state of DB is denoted as $db = I$, where $I \subseteq \mathbf{NCL}^{\Sigma}$. The closure of I under the Closed World Assumption is denoted as $\bar{I}$, ie $F(\bar{I}) = F(I) \cup \{\neg\alpha \mid \alpha \in \mathbf{AF} \setminus F(I),\ \alpha \text{ ground}\}$. A state $db = I$ is always consistent, viz $C \subseteq Th(\bar{I})$ holds.

Definition 7 Let $\chi(C)$ denote the set of all consistent data sets with regard to C, ie $\chi(C) = \{I \subseteq \mathbf{NCL} \mid C \subseteq Th(\bar{I})\}$, and let $db = I$ be the present state of DB. From a declarative viewpoint, a transaction τ is a set $X \subseteq \mathbf{NCL}$. From an operational viewpoint, a transaction $\tau = (\delta, \iota)$ alters the set I into $X \subseteq \mathbf{NCL}$, which we denote as $I \xrightarrow{\tau} X$; $\tau = (\delta, \iota)$ is completely characterised through two components: the set of facts deleted

[*] We omit the superscript whenever the respective signature is evident.

from I, δ, and the set of facts newly included into X, ι, ie $\delta \cap \iota = \varnothing$ and $F(I) \setminus \delta = F(X) \setminus \iota$. If $X \in \chi(C)$, then τ is accepted and $db = X$ is the new state of DB. The set of all accepted transactions with regard to $\chi(C)$ is denoted by $T(C)$, ie $T(C) = \{\tau(\delta,\iota) \mid I \xrightarrow{\tau} X, I, X \in \chi(C)\}$. ◉

Moreover, we assume that the only way to communicate with a LDB is through an interface with the following properties:

- LDB's response to the command LIST is a complete listing of Σ, C and I.
- LDB's response to a query is:

  Syntax error if the query is not a valid query in the language over Σ.

  otherwise, a possibly empty set of ground substitutions which define a subset of $F(I)$.

- LDB's response to a transaction τ is:

  Syntax error if τ contains a clause which is not in $\mathbf{NCL}^{\Sigma}$.

  Accepted if $\tau \in T(C)$.

  Rejected if $\tau \notin T(C)$.

- LDB's response to any other input is Unrecognised command.

2.3 Persons and rights

The database presented above is an open one because it cannot tell one user from another – it answers any query and follows any valid transaction in the same manner. A database must be able to recognise the users if it is expected to treat them differently. Therefore we add to our database a set P of all users or persons who have access to it. The rules of the database behaviour towards a user are usually laid down in relations. Relations expressing rights and prohibitions are of a special interest to us. We intro-

duce for each user $p \in P$ the following rights:

- $RS_p \subseteq \mathbf{CL}^{\Sigma_p}$* determines the clauses a person may see as an element of I or C.
- $RD_p \subseteq RS_p$ determines the clauses a person is allowed to delete.
- $RI_p \subseteq \mathbf{CL}^{\Sigma_p}$ determines the clauses a person is allowed to insert.

Now we have arrived at a database which recognises different users and is able to behave in accordance with the stated rights. We call it a database with rights.

2.4 Forms of secrecy

A secure logic database is a logic database with rights together with a set of confidentiality requirements. Speaking in a colloquial manner, a confidentiality requirement is a statement of the form:

d should be kept secret from p with regard to DB

where d is a component of LDB and p a user from P. A user knows that DB is a LDB. Thus only the elements which form a particular LDB-scheme or a LDB-state can possibly be kept secret: the symbols of the signature, the terms, the atoms, the clauses, the integrity constraints and the data.

2.4.1 Confidentiality of atomic formulae

Let us consider the confidentiality requirement:

$\alpha \in \mathbf{AF}^{\Sigma}$ should be kept secret from $p \in P$ with regard to DB

Spalka (1994a) has shown that the confidentiality of an atomic formula can be related to its derivability, ie to the membership-problem with respect to $Th(I)$. The following table illustrates the different possibilities of a relationship between an atomic formula and a set of formulae.

* For the moment it suffices to know that the sets of symbols of Σ_p, the signature of p, are subsets of the respective sets of Σ. The motivation for the removal of a symbol from Σ_p is given later.

DoC*	Answer	Amount of information on derivability	
		Informal	Formal
(G0)	Yes	Positive definite	$\alpha \in Th(\bar{I})$
G1	Maybe	Indefinite	$\alpha \vee \alpha' \vee \alpha'' \vee \ldots \in Th(\bar{I})$
G2	No	Negative definite	$\alpha \notin Th(\bar{I})$
G3	Don't know	Indeterminate	$\alpha \notin Th(\bar{I})$ and $\neg\alpha \notin Th(\bar{I})$
G4	Don't understand	No information	$\alpha \notin \mathbf{AF}$

The entries in the table are interpreted as the database's answer to the user's question 'Does $\alpha \in Th(\bar{I})$ hold?' in the situation when $\alpha \in Th(\bar{I})$ actually holds. There are five possible answers. The first answer obviously does not preserve the confidentiality of α, but which of the remaining four does? We see that it is not sufficient to merely require that α should be kept secret from p, in addition we must specify how much information on α with regard to I is p allowed to have.

At G4 the database pretends that α cannot be constructed in its language. It gives the user no information on the relationship between α and I. At G3 the database understands the user's question, but it pretends that it has insufficient information to answer it. At G2 the database's answer is a definite 'No'. Finally, at G1 the database tells the user that it knows the answer but it will not give it him. From the viewpoint of secrecy, G0 relates to facts which are not secret. G1 is the weakest and G4 the most stringent form of confidentiality.

From now on a confidentiality requirement for an atomic formula must be accompanied by a degree of confidentiality, ie it is of the form:

α should be kept secret from p with regard to I at the degree of confidentiality G

where G can be G1, G2, G3 or G4.

---

* Degree of Confidentiality

The interpretation of G4 in a LDB is easy since all we have to do is to remove one symbol from an user's signature he would need to construct the secret fact. However, it should be kept in mind that G4-requirements may unreasonably deteriorate the usefulness of the database, since the removal of one symbol from a user's signature reduces his language by a whole group of clauses.

G3 is trivially not satisfiable in a LDB with the CWA. The CWA tells us that for each atom α, either $\alpha \in F(\bar{I})$ or $\neg\alpha \in F(I)$ holds.

The interpretation of G2 is again simple. Here the CWA tells us that $\alpha \notin F(\bar{I}) \Rightarrow \neg\alpha \in F(I)$, ie the non-derivability of the secret fact implies by default the derivability of the negation.

In order to illustrate the interpretation of G1, let us consider the following example.

Example 1 Let $\Sigma = (\mathbf{FS} = \{a\}, \mathbf{PS} = \{p,q,s\})$ and $C = \{p(X) \vee q(X) \leftarrow s(X)\}$ be a LDB-scheme visible to a user u. Let moreover $F(I) = \{p(a), s(a)\}$, and $p(a)$ should be kept secret from u with regard to I at the degree of G1. Then $p(a)$ must not be derivable for u. Thus we reduce u's set of positive data to $F(I_u) = \{s(a)\}$. Now the trouble is that I_u does not satisfy C, and we owe the user an explanation. We suggest to tell him that:

- the integrity constraints in C are always satisfied by the data in I
- his data may seem to violate C due to some secrets

Now the user is able to identify the violated constraint, and through a simple substitution he can find out that $p(a) \vee q(a) \in Th(I)$ holds, viz either $p(a)$ or is $q(a)$ true. It may be $p(a)$ but may as well not be $p(a)$. •

We see that the interpretation of G1 in a LDB involves some interactions and new conventions. One can object that it is hardly acceptable if confidentiality depends only on two choices, eg $p(a)$ or $q(a)$. However, the integrity constraints as such are given by an

application and G1 only interprets them. Thus if the relevant integrity constraint is a definite clause, then it may be impossible to keep a fact secret; if on the other hand the constraint is a disjunction of 7000 links, then G1 seems to be fairly strong.

Although G1 is the weakest form of confidentiality, it has a number of important advantages over the other forms. Firstly, it does not lie to the user. It only provides him with a weaker information than it is capable of, but this information is still true. If we contemplate the possible consequences of a lie from a practical and ethical point of view, then it seems preferable to give imprecise rather than false information. This viewpoint is shared by many researchers.[*] Secondly, the enforcement of G1 can be achieved without cover stories, viz explicitly introduced lies. Being compared to it, G2 may require the maintenance of a consistent set of lies which greatly raises the effort needed to enforce it.

2.4.2 Confidentiality of clauses

Most previous approaches have defined the confidentiality of a clause as non-derivability, ie a clause φ is secret for a user p with regard to a set of clauses I, if p cannot conclude that $\varphi \in Th(\bar{I})$ holds. We believe that this definition is inappropriate. Let us consider two motivating examples.

Example 2 Let $db = I = \{r(a)\}$, then the clause $\varphi = r(X) \leftarrow q(X)$ is derivable from I. Yet there is no substitution which makes φ's body true; its derivability depends completely on the default data given by the CWA. In particular, φ remains derivable even if we substitute any other predicate symbol of the signature for q.

Example 3 Let $F(I) = \{q(a_1), r(a_1), \ldots, q(a_{1000}), r(a_{1000})\}$, then $\varphi = r(X) \leftarrow q(X)$ is derivable. In order to make φ non-derivable, it is already sufficient to adjust the data in I so that eg either $r(a_1)$ is deleted from $F(I)$, or $q(a_{1001})$ is inserted into $F(I)$.

---

[*] CSFW (1994).

In the first example, the clause is trivially derivable, but this information seems to be of little avail. In the second example, the clause is no longer derivable as soon as there is just one substitution which makes the clause's body true while leaving its head false. But can we just neglect the 1000 other substitutions where this is not the case? When we say that a clause is confidential, do we just regard the clause as any other formula, or do we implicitly express that a confidential rule derives confidential data? There are presumably no formal grounds on which this question can be answered. However, backed by our motivation, we believe that the second part of this question should be answered with 'Yes'.

We therefore give the following definition. Let φ be $\varphi = \alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$. The confidentiality requirement

φ should be kept secret from the user u with regard to I at the degree of G

is interpreted in a LDB as follows:

i) $\varphi \in I_u$ and $\varphi \notin C_u$

ii) For all ground substitutions $\pi(\beta_1 \wedge \ldots \wedge \beta_n)$ so that $\pi(\beta_1 \wedge \ldots \wedge \beta_n) \in Th(I)$, it is required that $\pi(\alpha_1 \vee \ldots \vee \alpha_m)$ should be kept secret from p with regard to I at the degree G.

Analogous to the previous section, G4-secrecy of a clause is linked to the user's signature, and a clause cannot be kept secret at the degree G3 due to the CWA. The G2-degree means that $\pi(\alpha_1 \vee \ldots \vee \alpha_m) \notin Th(I)$. Since $\pi(\alpha_1 \vee \ldots \vee \alpha_m)$ is a disjunction of ground atoms, this can only be satisfied when each link of the disjunction is not derivable. The G1 degree does not seem to make sense for indefinite clauses. However, G1 has a simple interpretation for definite clauses, eg $\varphi = \alpha \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$:

- $\varphi \in I_u$ and $\varphi \notin C_u$

- all ground instances of the head derivable from the present state's data, ie $\pi(\alpha) \notin F(I)$, should be kept secret at the degree G1.

Finally, we note that the given interpretation of a clause's confidentiality is a proper extension of an atomic formula's confidentiality. If the clause is definite and has an empty body, then its head is a ground atom.

2.4.3 Some remarks

A complete discussion of the confidentiality of symbols of the signature and of terms can be found in Spalka (1994a).

In our opinion, one of the main advantages of the presented formalisation is that it is no longer necessary to use ambiguous, colloquial descriptions of confidentiality-degrees, eg fully secret, disclosure of the existence, partial disclosure of secret information. To give an example, when is an atomic formula $r(t_1,\ldots,t_k)$ partially disclosed to a user?

- When the user knows that the symbol r or some terms t_i are elements of the database-language?
- When the user knows that $r(t_1,\ldots,t_k)$ is a valid formula, viz the database recognises it?
- When the user knows that some atomic formula comprising t_i is derivable?

The degrees G1-G4 avoid this confusion.

This paper investigates only the enforcement of G1. It is the weakest form of confidentiality and we do not say that it is suitable for all situations. Yet we feel that it may be appropriate for some situations and we know that its investigation is a necessary preliminary step for the investigation of G2, which seems to be the commonly preferred form of confidentiality.

2.5 Personal database profiles

Let DB be a database with the scheme $DB = (\Sigma, C)$ and the state $db = I$. The application of RS, the right to see, to DB provides for each user p his profile $DB_p = (\Sigma_p, C_p)$ and $db_p = I_p$. One of the requirements to the profile is that it satisfies the confidentiality requirements for the user p. But there is more than this. Our starting point has been an open database. Then we have added users and rights to it. If a user possesses all rights, then his profile is identical to the whole database. Otherwise, his profile is different from it. Should the database semantics of the whole database or of a profile be allowed to vary depending on the actual settings of the rights? We maintain that the desirable answer is in both cases 'No'. We would like to look on a profile as an independent open database which respects the validity of the whole database. We state four expectations about the semantics of a database with rights:

(i) The original database $DB = (\Sigma, C)$ and $db = I$ is valid if $C \subseteq Th(I)$.

(ii) A profile $DB_p = (\Sigma_p, C_p)$ and $db_p = I_p$ is valid if $C_p \subseteq Th(I_p)$.

(iii) The validity of a profile is subordinate to the validity of the original database:

$$I_p = Th(I_p) \wedge C_p \subseteq Th(I) \Rightarrow C \subseteq Th(I)$$

(iv) The validity of two different profiles is independent from each other.

Points (i) and (ii) restate the fundamental definition of integrity in databases. To question anyone of them means to question integrity constraints as such, viz we would no longer talk about databases. Point (iii) allows for the rejection of an update command issued by p even if the resulting state $db_p = I_p$ is valid. This can be the case when C_p contains stronger properties than C. This is consistent with our formalism and can be useful in practice. These considerations show that it no longer makes sense to ask if a

---

Point (iii) expresses also that the integrity of non-confidential data implies the integrity of confidential data. This may sound strange at first. However, the whole database comprises all data and its integrity may never be violated – this is a fundamental property of a database.

database with rights is valid when we have the definition of validity of an open database in mind. We therefore give a new definition of the validity of a database with rights *DB*.

We still say that *DB* is valid, if the state $db = I$ is valid. But we say that *DB* is locally valid for a $p \in P$ if DB_p is valid, or simply that *DB* is locally valid if it holds for all profiles, and we say that *DB* is globally valid if $db = I$ is valid and *DB* is locally valid, ie all profiles are also valid.

In this light the notion of global validity of a database with rights seems to be the matching counterpart to the notion of validity of an open database.

Point (iii) requires some final explanation. Animated by other authors, we have also contemplated the situation where a command is rejected because $C \not\subset Th(I)$, even if $C_p \subseteq Th(I_p)$ would hold. We exclude it for the following reasons:

- The user p has been granted his rights on condition that he is trusted to make use of them. To us it seems judicious to provide him with an explanation for the acceptance as well as for a rejection of his actions authorised by these rights.
- We have considerable doubt whether it makes sense at all to talk of a database from the user's point of view when the part of the database seen by him exhibits random behaviour. We could then omit C_p completely from his profile, since he would never know if a decision made by C_p is not overruled by some *invisible* authority.
- Finally, the formalism the database is based on would be of no use for the determination of the risks of disclosure. An autonomous profile gives the user no opportunity of finding out any properties of C. In the other case the database would not have the slightest idea of the information which the user already has deduced and will deduce from its behaviour.

2.6 External assumptions

Although we have taken a theoretical approach to security, it would be inappropriate to neglect the real context of an environment where a database may be used. There are three main areas of concern outside the database:

- The information stored in the database a user has already known of.
- The process of rights-granting.
- Software and hardware manipulations.

2.6.1 Individual knowledge

When we talk about a database with users who are real persons, we must take their individual knowledge relevant to the database contents into account. The necessity to model the user-knowledge has been recently strongly emphasised by Landwehr and LaPadula. A person's own knowledge is evidently not subject to the allocation of rights in the database. Thus it can be considered as the lower bound on the size of a person's database profile.

We denote the properties of I known to p by E_p, we call E_p the expectations of p with regard to the contents of I_p. In order that the database is able to take E_p into consideration, its contents must lay within the expressive power of the database. Hence we assume that $E_p \subseteq \mathbf{CL}$. Moreover we assume that p's perception conforms to the notion of validity as it is understood by the database, i.e. $\forall p \in P: C \subseteq Th(I) \rightarrow E_p \subseteq Th(I)$. Note that C is also sufficient to express p's knowledge of any piece of information from I. In this paper we assume that an E_p is given for each p.[1]

© SpW (1994)

[1] Along with it we can also introduce lower bounds on the size of the signature and the set of clauses. This, however, is not relevant to this paper.

We are well aware of the difficulty in determining the expectations of a user in practice and it is obviously out of question that safe clauses can only capture a part of the relevant user knowledge, but still we

2.6.2 Trustworthiness

Our database recognises rights, but the decision to grant or refuse a right is made outside it. Who makes the decision and what are his criteria? The first part of the question is for sure of great practical importance, but it has no relevance whatever to the problem of confidentiality when according to the criteria only *reasonable* decisions are approved of. The motivation for the criteria is given by Landwehr (1981):

> When a document is not in a safe, it is in the custody of some individual trusted not to distribute it improperly.

This means that a person is only granted a right if he can be trusted to comply with its intended use. In particular, we may assume that the refusal of a right to a person within the database will not be circumvented outside the database by other persons possessing this right. When the right to see is considered, a specific confidentiality is usually assigned to the document, ie the object of protection, and an individual trustworthiness to each person. The condition that the person's trustworthiness is adequate to the object's confidentiality by some kind of measurement, is necessary but not sufficient for the granting of this right.

Our database with rights makes therefore the following assumptions:

- The granting of a right to a person is always justified by this person's trustworthiness which is established outside the database.
- A person always behaves in accordance with the expectations implied by his trustworthiness.

---

believe that it is a good point to start with.

Landwehr (1981):250.

One should note that this assumptions do not imply that a person is trusted to see confidential data. If this person is able to trick another person to obtain confidential information, then there is nothing the database can do about it.

2.6.3 Others

The security provided by a database is just one component of an overall security policy. We do not consider any software attacks like Trojan Horse Programs, etc, nor any hardware attacks, like theft, wire tapping, etc. These issues are related to a particular implementation but not to our logical model. We take only those risks into account which can be expressed within our formalism, viz which the database can recognise independent of its actual implementation. We make the following assumptions:

- The only way for a person to get a piece of information stored in the database is through the database interface intended for communication with this person.
- Any piece of information a person may otherwise get is in tune with the database's present assignment of rights.

3 Enforcement of G1-secrecy

This paper concentrates on the enforcement of G1 only in logic databases. In this section we investigate the aspects of the various forms of reasoning about the interactions between a user and a database with regard to G1-secrecy. The primary purpose of this section is to enable the database to recognise when a secret fact may possibly be disclosed to a user. For this reason, our investigations do not extend beyond the expressive power of our database.

Let G_p denote the set of facts which should be kept secret from the user p with regard to $db = I$ at the G1-degree.

3.1 G1 and the right to see

We commence by considering of the case in which the user p is only an observer of his profile, that is, his rights *RD* and *RI* are empty. In his profile $DB_p = (\Sigma_p, C_p)$, we set

$C_p = E_p$ because C_p is not involved in the process of answering p's queries, and p is not allowed to issue any update commands.

First of all, as a formalisation of the external assumptions, we may assume that no secret fact is already known to p.* Secondly, we also assume that no secret fact is among $F(I_p)$. Otherwise the database will tell p a secret whenever he likes to ask about it. Finally, the user's state $db_p = I_p$ should be productively useful, ie it should contain exactly the information which the user needs to do his job. Now we face the problem that such a state may be invalid. We have to consider that E_p places a lower limit on the size of $db_p = I_p$ and we may not adjust E_p's contents. Thus an I_p which does not tell p any secret may well be too small for E_p. There are three possibilities to handle this situation:

i) We can try to enlarge $db_p = I_p$ to a valid set by including some clauses of $db = I$ which need not be kept secret from p and which guarantee that $F(\bar{I}_p)$ still does not comprise any secret.

ii) We can enlarge $db_p = I_p$ into a valid set if we add all the clauses primarily responsible for its invalidity to it.

iii) We do not adjust $db_p = I_p$. Instead we tell p that his current state is invalid due to some secrets.

The first possibility is obviously the most desirable one, but its application depends on the contents of the present state $db = I$. If there are no suitable clauses or we are not willing to show p clauses of which he has no need to know, then point (ii) suggests to relinquish some secrecy – this is rather a surrender than a solution. We believe that whenever point (i) cannot be applied, we should follow point (iii) and see whether this preserves G1-secrecy. To admit point (iii), we say that a profile DB_p is weakly consistent* if there is a subset $S \subseteq G_p$ so that $C_p \subseteq Th(\overline{I_p \cup S})$. Similarly, we say that a data-

* If this is not the case, then some external security precautions have failed to keep it back from him.

* Sometimes we call a valid state strictly consistent in order to draw a distinction to weak consistency.

base with rights is globally weakly consistent if there is a weakly consistent profile. One should note that we never allow DB to become inconsistent. Inconsistency is only tolerated as a local phenomenon of a profile due to the confidentiality requirements.

If DB_p is valid, then – from the database's point of view – secrecy is preserved. What can p find out if his profile is only weakly consistent? He can identify the violated integrity constraints of C_p and then look for the reason of the violation. Let us assume that $\alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n \in C_p$ is a weakly consistent constraint, ie it is false due to a secret fact. It can only be violated if all β_i are true and all α_i are false, viz there is a substitution π so that $\pi(\beta_1 \wedge \ldots \wedge \beta_n) \in Th(\bar{I}_p)$ and $\pi(\alpha_1 \vee \ldots \vee \alpha_m) \notin Th(\bar{I}_p)$. However, weak consistency tells the user that $\pi(\alpha_1 \vee \ldots \vee \alpha_m) \in Th(I)$. Firstly, we see that weak consistency is concordant with the definition of G1. Secondly, we see that a definite integrity constraint is not able to preserve secrets.[1]

When the user is left with $\pi(\alpha_1 \vee \ldots \vee \alpha_m)$, $m > 1$, as the result of his search, then each $\pi(\alpha_i)$ represents an equally well suited candidate for a secret. What else can he do to reduce the number of candidates? Principally, he could check his rights RS and find out that there is just one candidate which would not be visible to him. To close this gap, we assume that a user's rights manifest themselves only through the interaction with his profile. The only remaining way (within the database) for the user is to simulate insert commands[2] for each candidate. To keep the secret, there must be at least one more candidate the insertion of which would lead to a consistent profile.[3] This method can preserve the secret fact which is responsible for the weak consistency of the constraint, yet it may disclose another secret.

---

[1] Note that if there is more than one substitution like π, then each one identifies a secret fact since otherwise DB_p would not be weakly consistent..

[2] Since p has no insert-right, he is forced to perform these *what-if* operations outside the real database but they are still within the expressive power of our formalism.

[3] cf Example 1, p 11.

Example 4 Let $C_p = \{r(X) \vee s(X) \leftarrow q(X)\}$, $I_p = \{q(a), v(X) \leftarrow r(X), v(X) \leftarrow s(X)\}$, and $G_p = \{r(a), v(a)\}$. This profile is weakly consistent and preserves G1-secrecy of $r(a)$, yet the secret fact $v(a)$ is disclosed.

Definition 8 Let $\varphi = \alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$ be a weakly consistent constraint. $\pi(\alpha_i)$ is a candidate for a secret if $I_p \cup \pi(\alpha_i)$ is consistent. φ is G1-secrecy-preserving if there are at least two candidates for a secret and there is no secret fact d such that $d \in F(I_p \cup \{\pi(\alpha_i)\})$ holds for all candidates $\pi(\alpha_i)$.

Each violated but G1-secrecy-preserving constraint can be separately considered. Let $\gamma_1, \ldots, \gamma_k$ be these constraints and let K_{γ_i} denote the disjunction of the candidates for a secret. Weak consistency due to the simultaneous violation of all γ_i tells p only that $K_{\gamma_1} \wedge \ldots \wedge K_{\gamma_k} \in Th(I)$.

Lemma 1 A profile DB_p is G1-secrecy-preserving for RS if:

- No secret clause is among C_p or I_p
- No secret fact can be derived from I_p
- DB_p is consistent, or it is weakly consistent and each violated integrity constraint is G1-secrecy-preserving.

3.2 G1 and the right to delete

Let DB_p be a G1-secrecy-preserving profile for RS. We now assume that p has the right to delete some data. Let $\varphi \in RD_p$; we assume that the state $db'_p = I'_p$, consequent upon the acceptance[1] of the *DELETE* φ command, does not contain φ, and we denote by δ_φ the set of atoms which have disappeared from $F(I_p)$ together with φ, viz $\delta_\varphi = F(I_p) \setminus F(I'_p)$. This implies that a secret fact which has not been among $F(I_p)$, is not among $F(I'_p)$ either. We therefore concentrate on the integrity constraints in the

[1] Note that a delete command is accepted in contravention of C_p if the new state is weakly consistent.

consideration of the following possible cases:

	DB_p	$DELETE\ \varphi$	DB'_p
1.	strict	accepted	strict
2.	strict	accepted	weak
3.	strict	rejected	(no change)
4.	weak	accepted	strict
5.	weak	accepted	weak
6.	weak	rejected	(no change)

(1) DB_p has been secrecy-preserving, so is DB'_p, since all constraints are satisfied.

(2) Let $\gamma = \alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$ be one of the violated constraints of C_p. The fact that γ has been valid in DB_p but is not in DB'_p (while it retains validity in DB') means that there is a substitution π so that some $\pi(\alpha_i) \in \delta_\varphi$.[1] Thus γ is only secrecy-preserving if K_γ with all its links in δ_φ removed, is also secrecy-preserving. This point diverts our attention to the fact that to handle update commands we must maintain a history starting at a particular state. All atoms from γ's head p knows to have deleted can no longer belong to K_γ.

Example. Let $C_p = \{s(X) \vee r(X) \leftarrow q(X)\}$, $I_p = \{q(a), r(a)\}$ and $G_p = \{s(a)\}$. After the user has deleted $r(a)$, the profile is weakly consistent, but the secret is disclosed.

(3)÷(6) According to our definition of a profile DB_p as an independent database which behaves in conformity with DB, the command is rejected since $C_p \not\subset Th(I'_p)$ and no secrets are involved in this decision.[2]

(4) This case is secrecy-preserving. Here a previously weakly consistent constraint

[1] The constraint γ is an indefinite clause, for otherwise the delete command would have been rejected.

[2] Otherwise the command would lead into a weakly consistent state.

is now strictly consistent. If the set of positive ground facts has become smaller, this can only happen when an instance of an atom of the constraint's body has been deleted.

Example 6 Let $C_p = \{s(X) \vee r(X) \leftarrow q(X)\}$, $I_p = \{q(a)\}$ and $G_p = \{s(a)\}$. After the user has deleted $q(a)$, the profile is strictly consistent.

(5) A weakly consistent constraint's property of being secrecy-preserving is not affected by delete commands since the sets K_p, once they are established, are invariant with respect to them.

Example 7 Let $C_p = \{v(X) \vee s(X) \vee r(X) \leftarrow q(X)\}$, $I_p = \{q(a), r(a)\}$ and $G_p = \{s(a)\}$. After the user has deleted $r(a)$, the profile is weakly consistent. The candidates for a secret are $r(a)$ and $s(a)$. This situation is now independent of any other delete operations.

Lemma 2 A profile DB_p is G1-secrecy-preserving for RD if:

- DB_p preserves G1-secrecy for RS
- Any newly created set of candidates for a secret has more than one member.

3.3 G1 and the right to insert

Let DB_p be G1-secrecy-preserving for RD and $RI_p = \mathbf{CL}^{\lambda_p}$. Let $\varphi \in RI_p$; we assume that the state $db'_p = I'_p$ consequent upon the acceptance of the *INSERT* φ command contains φ and we denote by ι_φ the set of atoms that have been newly included in $F(I_p)$ together with φ, ie $\iota_\varphi = F(I'_p) \setminus F(I_p)$.

If $\iota_\varphi \cap G_p = \varnothing$, then the new clause does not produce any new facts which should be kept secret from p. Hence the data do not disclose any secret. This is always the case when the user behaves concordant with his trustworthiness.

However, the user can, eg by mistake or on purpose, insert a clause so that $\iota_\varphi \cap G_p \neq \varnothing$. We take the view that in this case the reaction of the database should be guided by the

external assumptions, ie p has not gained knowledge of any secret. The database may not simply enter φ into RS_p since this means admitting that the secret is disclosed. We believe the database should assume that the user has created his own φ and it must find a way to resolve the name clash which has now arisen in the global name space of the data in I. One possible solution is the globalisation of φ in I by qualifying it with the name of p. Let $\varphi = s(c_1,\ldots,c_k)\leftarrow\ldots$, then it is sufficient to qualify the head. In particular we can either qualify the predicate symbol itself, ie turn s into $s.p$, or we can qualify the terms of s, ie turn $s(c_1,\ldots,c_k)\leftarrow\ldots$ into $s(c_1.p,\ldots,c_k.p)\leftarrow\ldots$ This process must be transparent to p. The application of this method results in two different clauses φ and $\varphi.p$: φ is used in derivations initiated by persons who have the right to see it, and $\varphi.p$ is used as φ in p's derivations.

Now that $\varphi.p$ is a member of I_p, does it make sense to make it visible to other persons as well, that is, can we treat $\varphi.p$ as an ordinary element of I? The answer should be 'Yes'. The basic database semantics tell us that $\varphi = \mathbf{TRUE} \Leftrightarrow \varphi \in Th(I)$. If p is just trying to trick the database, then he will issue a delete command immediately after the insert command, hoping that nobody has noticed his efforts. Whereas a clause which is true for p will remain in the database as long as he considers it true. Purely by chance this clause has been given a conflicting name. Thus we think it right to include p's φ as $s(c_1.p,\ldots,c_k.p)\leftarrow\ldots$ into I and treat it (with respect to all persons except p) as an ordinary clause. The main advantage of this solution is that it influences neither the semantics of DB nor of DB_p, and we do not need to introduce notions like: *is believed by...*, *is only true for...* etc. Moreover it preserves the secrecy of all K_p.

We now turn our attention to the role of integrity constraints in the possible state transitions.

	DB_p	$INSERT\ \varphi$	DB'_p
1.	strict	accepted	strict
2.	strict	accepted	weak
3.	strict	rejected	(no change)
4.	weak	accepted	strict
5.	weak	accepted	weak
6.	weak	rejected	(no change)

The cases (3) and (6) pose no risks, since C_p provides the reasons for the rejection. Case (1) does not introduce any new risks, either. In case (4), p must have inserted a link of a K_γ so that a previously weak consistent integrity constraint is now satisfied.

Cases (2) and (5) can be considered together. Let us assume that a constraint $\gamma = \alpha_1 \vee \ldots \vee \alpha_m \leftarrow \beta_1 \wedge \ldots \wedge \beta_n$ of C_p has been valid in I_p but is now, due to some new elements of ι_φ, weakly consistent. Thus there are some elements in ι_φ which have made the constraint's body true so that no matching instances of the head's atoms are present in $F(I'_p)$. Thus the insertion has created a new K_γ. Again, γ is secrecy-preserving if K_γ has more than one link.

Example 8 Let $C_p = \{s(X) \vee r(X) \leftarrow q(X) \wedge v(X)\}$, $I_p = \{q(a)\}$ and $G_p = \{s(a)\}$. The integrity constraint is now satisfied. After the user enters $v(a)$, it is weakly consistent, and since there are two candidates for secrets, it preserves G1-secrecy.

Lemma 3 A profile DB_p is G1-secrecy-preserving for RI if:

- DB_p preserves secrecy for G1 and RD
- Insertions valid with respect to C_p are handled along the presented guide-lines.
- Any newly created set of candidates for a secret has more than one member.

3.4 Final remarks

We would like to mention that we have also contemplated the use of a special term, for example a term *secret*, for our purposes. However, we have abandoned this idea because preliminary steps have already indicated that we are going to run into troubles similar to those of the *NULL* value.

Finally we note that G1-secrecy can be applied to multilevel secure databases. The details of it are presented in Spalka (1994b).

4 Conclusion

A logic database has served as the starting point for this paper. We have shown a way to add persons and rights to it, so that the semantics of this database is properly extended; the users have been assumed to possess their own knowledge. We have introduced four possible meanings of confidentiality; when asked about a confidential clause, the database can (informally) answer 'I don't understand', 'I don't know', 'No' or 'Maybe'. Three of these answers are applicable in the presence of the Closed World Assumption. The formal version of the last possibility, denoted by G1, is allowed to provide the user with indefinite information on a secret. In the following we have concentrated on the enforcement of G1.

As our main result, we have identified conditions which guarantee G1-secrecy in situations where the user has the right to submit queries, delete and insert commands. The distinguished role of integrity constraints has been given careful attention. We have demonstrated that there is no fundamental conflict between security and integrity, but instead the constraints' role as *boundary conditions* has become apparent throughout the investigation. The presented formalism is well-founded, and has been completely expressed within the limits of standard predicate logic.

Our next steps will be:

- to investigate the enforcement of G2-security in logic databases and to extend it to a multilevel security model
- to search for a feasible, practical way to construct user profiles.

References

Berson/Lunt (1987a)
Berson, Thomas A., and Teresa F. Lunt. 'Security Considerations for Knowledge-Based Systems'. *Third Expert Systems in Government Conference.* Reprint. 1987.

Berson/Lunt (1987b)
Berson, Thomas A., and Teresa F. Lunt. 'Multilevel Security for Knowledge-Based Systems' *1987 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1987. pp 235-242.

Bonatti/Kraus/Subrahmanian (1992)
Bonatti, Piero, Sarit Kraus, and V. S. Subrahmanian. 'Declarative Foundations of Secure Deductive Databases'. Ed Joachim Biskup, and Richard Hull. *4th International Conference on Database Theory - ICDT'92.* LNCS vol 646. Berlin, Heidelberg: Springer-Verlag, 1992. pp 391-406.

Burns (1990)
Burns, Rae K. 'Integrity and Secrecy: Fundamental Conflicts in the Database Environment'. Ed Bhavani Thuraisingham. *3rd RADC Database Security Workshop 1990.* Bedford, Massachussets: Mitre, 1991. pp 37-40.

Cremers/Griefahn/Hinze (1993)
Cremers, Armin B., Ulrike Griefahn, and Ralf Hinze. *Deduktive Datenbanken.* Vieweg, 1993.

CSFW (1994)
Private communication. *IEEE Computer Security Foundations Workshop VII.* Franconia, New Hampshire, 1994

Cuppens/Yazdanian (1991)
Cuppens, Frédéric, and Kioumars Yazdanian. 'Logic Hints and Security in Relational Database'. Ed Carl E. Landwehr, and Sushil Jajodia. *Database Security V.* IFIP WG11.3 Workshop on Database Security 1991. Amsterdam: North-Holland, 1992. pp 227-238.

Gallaire/Minker/Nicholas (1984)
Gallaire, Hervé, Jack Minker, and Jean-Marie Nicholas. 'Logic and Databases: A Deductive Approach'. *ACM Computing Surveys* 16.2 (1984):153-185.

Garvey/Lunt (1990)
Garvey, Thomas D., and Teresa F. Lunt. 'Multilevel Security for Knowledge Based Systems'. *6th Annual Computer Security Applications Conference.* IEEE Computer Society Press, 1990.

Garvey/Lunt (1991)
Garvey, Thomas D., and Teresa F. Lunt. *Multilevel Security for Knowledge Based Systems.* Technical Report SRI-CSL-91-01. Menlo Park, CA: SRI International, 1991.

Garvey et al (1992)
Garvey, Thomas D., Teresa F. Lunt, Xiaolei Qian, and Mark E. Stickel. 'Toward a tool to detect and eliminate inference problems in the design of multilevel databases'. Ed Bhavani Thuraisingham, and Carl E. Landwehr. *Database Security VI.* IFIP WG11.3 Workshop on Database Security 1992. Amsterdam: North-Holland, 1993. pp 149-167.

Gougen/Meseguer (1984)
Gougen, Joseph A., and José Meseguer. 'Unwinding and Inference Control'. *1984 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1984. pp 75-86.

Landwehr (1981)
Landwehr, Carl E. 'Formal Models for Computer Security'. *ACM Computing Surveys* 13.3 (1981):247-278.

Lunt/Millen (1989)
Lunt, Teresa F., and Jonathan K. Millen. *Secure Knowledge-Based Systems.* Technical Report SRI-CSL-90-04. Menlo Park, CA: SRI International, 1989.

Meadows/Jajodia (1987)
Meadows, Catherine, and Sushil Jajodia. 'Integrity Versus Security In Multilevel Secure Databases'. Ed Carl E. Landwehr. *Database Security.* IFIP WG11.3 Workshop on Database Security 1987. Amsterdam: North-Holland, 1988. pp 89-101.

Morgenstern (1987)
Morgenstern, Matthew. 'Security and Inference in Multilevel Database and Knowledge-Base Systems'. *1987 ACM SIGMOD Conference / SIGMOD Record* 16.3 (1987):357-373.

Reiter (1978)
Reiter, Raymond. 'On closed world databases'. Ed Hervé Gallaire, and Jack Minker. *Logic and Data Bases.* New York: Plenum, 1978. pp 55-76.

Reiter (1984)
Reiter, Raymond. 'Towards a Logical Reconstruction of Relational Database Theory'. Ed Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt. *On Conceptual Modeling.* New York: Springer, 1984. pp 191-238.

Spalka (1994a)
Spalka, Adrian. 'Formal Semantics of Rights and Confidentiality in Definite Deductive Databases'. *IEEE Computer Security Foundations Workshop VII.* IEEE Computer Society Press, 1994. pp 47-58.

Spalka (1994b)
Spalka, Adrian. 'Formal Semantics of Confidentiality in Multilevel Logic Databases'. *ACM SIGSAC New Security Paradigms Workshop 1994.* ACM Press, 1994.

Steinke (1991)
Steinke, G. 'Towards a Strategy for Achieving Security and Multi-User Integrity in Knowledge Based Systems'. *2nd International Workshop on the Deductive Approach to Information Systems and Databases.* 1991. pp 128-148.

Tener (1991)

Tener, William T. 'Knowledge Based Systems: Audit, Security and Validation Issues'. *7th International Conference on Information Security.* North-Holland, 1991. pp 111-121.

Thuraisingham (1991)

Thuraisingham, Bhavani. 'A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Data/Knowledge Base Management Systems'. *IEEE Computer Security Foundations Workshop IV.* IEEE Computer Society Press, 1991. pp 127-138.

Williams (1992)

Williams, James G. 'A Shift in Security Modeling Paradigms'. *1992-1993 ACM SIGSAC New Security Paradigms Workshop.* IEEE Computer Society Press, 1993. pp 57-61.

Wiseman (1990)

Wiseman, Simon. 'The Control of Integrity in Databases'. Ed Sushil Jajodia, and Carl E. Landwehr. *Database Security IV.* IFIP WG11.3 Workshop on Database Security 1990. Amsterdam: North-Holland, 1991. pp 191-203.

Wiseman (1991)

Wiseman, Simon. 'The conflict between confidentiality and integrity'. *IEEE Computer Security Foundations Workshop IV.* IEEE Computer Society Press, 1991. pp 241-242.

Access control and application design:

Chair: T. C. Ting

Uni. Conneticut, CT

A Fine-grained Access Control Model for Object-Oriented DBMSs

Arnon Rosenthal[a], James Williams[a], William Herndon[b], and Bhavani Thuraisingham[a]

[a]The MITRE Corporation, 202 Burlington Road, Bedford MA 01730, USA
[b]The MITRE Corporation, 7525 Colshire Drive, McLean VA 22102, USA
email: {arnie, jgw, wherndon, thura}@mitre.org

ABSTRACT

We have developed an object model for MLS object-oriented databases with per-element access control. We argue that an MLS object model should specify structures and operations supported by a trusted kernel and hence should be kept simple. "Convenience" operators and enforcement of noncritical integrity guidelines are provided by nonassured layers above the kernel. Also for simplicity, there is a single kind of labeled entity, the data element; these labels determine the protection for several other types of constructs. We identify assumptions in some prior models that conflict with commercial OODBMS requirements for language, compatibility, and performance. Finally, we explore some conflicts between flexibility of deletion and avoidance of polyinstantiation. Companion papers will address metadata, polyinstantiation, and ordered collections.

1. INTRODUCTION

Object-oriented database management systems (OODBMSs) are gaining popularity due to their inherent ability to represent conceptual entities as objects, paralleling the way humans view the world. This power of representation has led to a new generation of object database managers that can support applications such as computer aided design and computer aided management (CAD/CAM), multimedia information processing, artificial intelligence, and process control. For secure applications, this paper specifies an object model for securing multilevel secure (MLS) information.

1.1. The Need For A New Model

The model presented here is not radically different from other models that label individual elements. However, it appears that no published model provides all of the following:

- *Model Flexibility*: Some models impose restrictions to rule out states that appear unreachable or unnecessary. Paradoxically, such restrictions may complicate the

model and implementation, as they must be documented and enforced, and will be very difficult to remove later.

- *Element-level Access Control*: The need to split natural problem domain objects into objects each at a single security level greatly complicates application-building [Smit89 (sec. 5.2), Rose93]. Many of the existing models provide protection only on coarse granules (i.e., objects).

- *Consistency with Industry Trends*: Some existing models make method calls very expensive, forbid multiple inheritance, require that the language encapsulate individual objects, or require that the DBMS maintain the integrity of all stored references. These decisions are inconsistent with the languages (e.g., C++) and tradeoffs common in the OODBMS industry.

- *Explicit Operation Specifications*: Few existing models define their operation semantics in detail, especially for deletion and regrading. Thus it is difficult to judge whether an operation's results can be expressed by the model's structures, or whether features that impact operations at different levels, such as object existence and identity, can cause channels.

This paper addresses the above issues, and extracts lessons that may be of use with other models. A fuller model that also addresses metadata, polyinstantiation, and collections (e.g., sets, lists, trees) is presented in [Rose94].

1.2. Paper Overview

1.2.1. Role of a Secure Data Model

The specification of an MLS DBMS can be done at three layers—the *kernel* interface, which defines the allowable structures and basic trusted operations, *guidelines* that advise about structures that should be viewed with skepticism, and the *application programmer interface* (which includes convenient operators built above the kernel).

The model presented in this paper defines the kernel interface, taking care to minimize its size and yet provide appropriate flexibility for users. Kernel facilities must be present and assured so that they can be relied on in implementing other features. Kernel operations on a legal state always produce another legal state, even if privileged. In contrast, integrity constraints are frequently enforced only at transaction commit time. We spend little time on the application interface's additional "convenience" features, leaving them to DBMS designers.

Explicit guidelines are warnings about database states that seem dubious (e.g., dangling references, inaccessible data, and odd patterns of classification). Since such states do not compromise security, their detection need not be assured. In fact, the model is not violated if DBMS builders omit enforcement of some guidelines, or if an application overrides warnings, especially during intermediate steps in a larger process. Some integrity-related guidelines are discussed in the appendix.

1.2.2. Paper Organization

The Uniform Fine-grained Object Security (pronounced "U.F.O.s") family of models [Rose94] attempts to address all the difficulties mentioned in Section 1.1. This paper presents a limited model called Single-valued UFOS (sv-UFOS), which explores the possibilities and limitations of a model with neither polyinstantiation nor covert channels.

The model description begins in Section 2 with a simple nonsecure object model, above which richer type systems could be built. We show how a polyinstantiated data model has different natural security entities from one that forbids polyinstantiation. Section 3 discusses security issues in more detail and illustrates how various situations would be modeled. Section 4 presents operator semantics (including the handling of security levels) and examines alternative deletion operations. Section 5 covers design goals and implementation issues. And finally Section 6 addresses future work and conclusions. In addition, the appendix discusses integrity enforcement tradeoffs.

1.2.3. Comparison with Previous Work

Several MLS object models have been previously published. Some of them assume that an object contains information at only a single level, thereby forcing application programmers to decompose the application's natural conceptual objects [Keef89, Jajo90, Mill92, Bert94]. This decomposition is very onerous and single-level restrictions would be difficult to remove from a DBMS once implemented, so we consider the such restrictions unsuitable for the long term [Smit89, Rose93].

Sv-UFOS is closer to published object models that permit multilevel objects, such as [Gajn88, Thur91, Smit89, Morg90].[1] Unlike some relational work (especially SeaView, [Lunt94]), most of these MLS object models define operators' semantics informally; specifying sv-UFOS operators in detail caused us to find and fix many problems. Also, some of the above models include many constraints on entities' relative levels. The costs of such constraints are discussed in Section 5.

We protect several of the varieties of information identified in [Smit89], including associations to values, associations to abstract objects, values, objects, and labels. Section 3.2 illustrates how this is achieved by means of controls on read and write access to a single construct, the element. (Our operational semantics do make special provision for the existence and labels; however, we benefit in that operations defined on ordinary attributes can also manipulate existence information.)

Several models inspired by Smalltalk exploit *object encapsulation* and assume that private attributes of an object can be accessed only by methods called on that object [Jajo90, Bert94, Oliv94]. Such results appear very difficult to transfer to the mainstream of the OODBMS industry, where C++ is the predominant language. Three major difficulties are examined below

[1] [Bisk88] permits objects whose elements have different access classes, in a DAC setting.

First, and most seriously, C++ encapsulates classes rather than objects; that is a method on a C++ object o can access the private state of *any* object in *class*(o).[2] Second, some models assume that methods can write only in ways mediated by the OODBMS, but method code is usually allowed to invoke any system capability. Finally, for high-assurance systems, verifying that a language's encapsulation is enforced may require assuring a substantial part of the compiler.

2 THE UNDERLYING OBJECT MODEL

This section defines a simple, conventional object model, which we subsequently extend to an MLS object model. We also examine two candidates for the basic security entity—*element* (a pair <object instance, attribute>) and *association* (a pair, <element, value>).[3]

An *object instance* (or just *object*) consists of a state (stored data) that associates a value with each attribute, and a set of defined operations or *methods* that act upon that state. (Methods are discussed only briefly in this report.) Objects are instantiated from a *class* definition in which all of an object's attributes and methods are defined; an element's value must be class-appropriate. An object will often be denoted using an optional prefix ("a" or "the") plus a class name or a human-understandable instance name, as in *aShip*, *theNimitz*, or *Beijing*.

Objects are an abstraction: user programs and UFOS model operations refer directly only to *values*. There are two kinds of values: *primitives*, such as integers and strings, and *object references* (often referred to as *handles*). Given a handle, the DBMS can test if it references an object currently in the database, and if so can access that object. To avoid serious impact on performance and language compatibility (as discussed in the appendix), referential integrity is merely a guideline. If o denotes an object, @o denotes a value that can be used as a handle for o.

Given an object o and attribute A, $o.A$ denotes the corresponding element. An association assigns a value to a data element, the *value* of the association. For example, *theQE2.MaxSpeed* and *theQE2.Destination* might be associated respectively with the integer 40 and the handle @*Singapore*.

Several technicalities should be noted. First, an element's value and an operation's arguments are not objects; only such values are passed to user programs. Second, the current database state has at most one association, and hence one value, for each element. Thus we can use $o.A$ to refer to either an element or the association, if any, from that element. Third, there is no requirement for a unique object identifier that is shared across security levels—at the implementer's option, an object may have any number of handles. Finally, the same primitive

---

[2] C++ designers have traditionally emphasized software engineering and performance rather than security.

[3] Both the object-oriented and security communities use the term *object*. To avoid confusion, this report uses the term in the first sense, as an object possessing attributes and methods. The units of security are referred to as *entities*.

or handle may be the value of many elements, potentially requiring different security protections. For example, *@Beijing* may be the value associated with both *KissingerTrip.Destination* and *China.Capital.*

Operations create and delete objects and associations to values. For purposes of the abstract model, values are never created, deleted, or changed; conceptually, all possible values (integers, strings, handles) exist when the database is created. The implementation must support this illusion without violating security. The link between a handle and an object's physical storage is maintained by the DBMS, transparent to the user.

2.1. Basic Security Entities

In sv-UFOS the basic security entity is the element; in polyinstantiated-UFOS (p-UFOS) it is the association [Rose94]. The rationale for the different choices is discussed here.

One way to understand a conventional database is that each element identifies a property of some real-world object, and each association asserts the current state of such a property. An association is thus the basic unit of fact. A common notion of polyinstantiated database is to allow a separate assertion about such a property at each security level; some levels may make no assertion. When associations are the protected entities, a user at level L knows nothing about associations at any level L' dominating or incomparable to L. An update to an element that already has an association at L' cannot be refused without creating a covert channel.

To avoid polyinstantiation, we associate a security level with *each* element, to govern the sensitivity of the association it holds. Elements' security levels are readable by any user whose level that can see that the object exists; thus each update operation can test whether an update is permitted, even if it cannot see the element's association and value. The model includes operations to change an element's level.

Neither values nor elements' levels are "first class" security entities. Values always exist in principle; the sensitivity is attached to their usage, as shown above for *@Beijing*. Elements are rather closer to being entities, but they do not require their own labels.

3. STRUCTURES, CLASSIFICATION, AND SUBJECT LEVELS

This section defines the constructs of the secure object model. Section 3.1 describes the security entities of the model, and their usage is illustrated in Section 3.2. Section 3.3 sketches the model's operations (which are presented in detail in Section 4).

3.1. Security Levels of Elements

Security entities are those for which access classes can be individually assigned. To simplify the theory and (we hope) the DBMS implementation, the *only* kind of security entity is the element. The sensitivity of each element $o.A$ is indicated by a classification, called $level(o.A)$. Levels typically include hierarchical and nonhierarchical components and are partially ordered. Throughout the discussion, the level of the current user program is denoted L_S.

Object existence is a security entity, as in other models, and operations need to test whether an object's existence level is dominated by L_S. Rather than define a completely new

model construct, we represent existence using an additional Boolean attribute, *exist*, in each object. Most kernel and higher level features applicable to ordinary elements also apply to existence.[4] These features include labels, query languages, auditing of updates, triggers, and (in p-UFOS) polyinstantiation to reflect disputes about an object's existence. We impose the constraint that for every element *o.A*, $level(o.A) \geq level(o.exist)$. Element levels may be stored in each object, along with *exist*. Alternatively, if all objects in a class have the same level for some attribute, the schema can identify that level.

As in MLS relational DBMSs, a security subject corresponds to a process or executing program, which is assumed to be single-level. Data residing in a process's memory is part of the process. Method invocations and attribute references execute as part of a program, like ordinary procedure calls; they *do not* cause creation of a new subject.[5] The OODBMS permits access to the database only for the operations that are explicitly in the secure object model.

Our security policy for whether an unprivileged subject can read or modify a particular element is in the tradition of Bell and LaPadula. An unprivileged subject at L_S can retrieve the associated value of elements at or below L_S and can modify element values at or above L_S. The levels of elements of the form *o.A* are protected as if they were part of existence information at $level(o.exist)$, and furthermore may be updated only by special operations in the model (Section 4.3).

3.2. Representing an MLS Database in UFOS: Examples and Discussion

We now illustrate how the model's access controls for elements serve to protect other information of concern, such as existence, object identifiers (handles), and the fact that a value appears in the database. Any level that dominates $level(o.exist)$ may appear on any element.

Figure 3.1 shows two instances of the class *Person*, *p23* and *p78*, and one of class *Department*, *d123*. The existence of object *p23* is unclassified, as is *p23.p_name*. The associations from *p23.address* to "Times Sq" and of *p23.department* to the handle *@d123* are Secret. *p78* has a completely different pattern of security levels. Existence of *p78* is confidential, and handles of *p78* will appear invalid to unclassified operations. In this example, both Person records reference the same department, using associations to same handle *@d123*. However, *p78*'s association with the department is less sensitive.

---

[4] Update operators behave differently on *exist* than on other elements. If the OODBMS is implemented in an object oriented language, *Existence_Element* can be a subtype of type *Element*. *Existence_Element* can inherit most operations from *Element*, but some (e.g., Modify) will be overridden.

[5] Because object managers often handle both main-memory and (disk-based) persistent objects transparently, performance could suffer if all method calls were significant operating system or security events. We are not aware of any commercial object oriented programming language or DBMS that implements each object or each method call as a separate process. Such implementations seem appropriate only in an object model devoted to large-granule distributed computing. Even for Smalltalk, whose conceptual model consists of messages, the implementation typically approximates late binding for procedure calls.

p23

p_name	address	birthdate	department	*exist*
<u>	<s>	<u>	<s>	<u>
"Bill"	"Times Sq."	–	@d123	*true*

p78

p_name	address	birthdate	department	*exist*
<s>	<ts>	<s>	<c>	<c>
"Max"	"Old Town"	"1/1/71"	@d123	*true*

d123

d_name	phone#	budget	*exist*
<u>	<c>	<ts>	<u>
"Special Ops"	"555-1234"	10,000,000	*true*

Figure 3-1 Objects with Labeled Elements

3.3. Operations in the Model: Overview and Illustrations

This section sketches the model's operations and discusses the kinds of information that are protected from access.

Operations in the UFOS model are procedures, callable from either methods or ordinary application programs. As procedures, the model's operations have input and output parameters that are bound to arguments in the calling program (i.e., to program variables that store or receive values). Issues of name scope and encapsulation would be part of the programming language but are not considered part of the secure object model; security of the system does not rely on compiler enforcement of object encapsulation.

The model's operations for accessing elements are *retrieve* and *modify*, and (expressed as a reference to the *exist* attribute) *effectively_exists*. The operations on whole objects are *create* and *delete*. Operations to raise and lower element levels are also provided. Like the majority of commercial OODBMSs, we provide no explicit operations or protection for k-ary relationships, except as ordinary objects. We now illustrate some operations to show how they protect various forms of information

Retrieval of *p23.department* returns the handle *@d123*. The association to the handle is protected at the Secret level, thus protecting the fact that *Bill* works in *d123*, and also protecting

information that the implementation might store in the handle.[6] The model does not release object identifiers to user. Thus far, the user has access only to the handle; neither the object identifier, the object itself, nor any of the object's element values has been retrieved. To access the referenced department's element values, one invokes *retrieve* with *@d123* as an argument; the new invocation is again subject to the security policy. The only way to determine that a value such as *"555-1234"* or *@d123* appears in the database is by retrieving it as the value of an element.

The example schema illustrates that UFOS permits objects that are visible but unreachable at some security level. For example, *d123* has Unclassified *d_name*, but there is no Unclassified reference to *d123*. This situation can arise in several ways, e.g., through deletion of a reference or through *Regrade*. As in single-level databases, an object can become totally unreachable. Like most programming languages, we handle such unreachable objects via garbage collection rather than by constraints enforced on each operation. Tradeoffs for integrity enforcement are discussed further in the appendix.

4. OPERATIONS OF SV-UFOS

This section describes the operations of the single-valued UFOS model and illustrates their use. The model includes operations on a single object, on the association of a single element, and on the level of a single element. The operations appear not to have channels, but we have not given a formal proof. Section 4.4 discusses alternatives to the potentially clumsy deletion operation; we show that they require relaxing the assumption that every object has a unique lowest level.

The operation interfaces were chosen for clarity in expressing the model; the application program interface can provide a layer of syntactic sugar. For example, to be compatible with legacy code that expects only a value, other output parameters (e.g., return code, element's level) might be returned by separate calls. This syntactic layer might be different for each programming language from which the OODBMS can be invoked and is not specified here. For each operation, we give a shorthand description in terms of objects, define the interface, and then give the semantics (including return codes to indicate the situation encountered).

If h denotes a handle, $\uparrow h$ will denote the object referenced by h. As a convention, we consider $\uparrow h$ to be a way to identify the "real" operand of the operation, an object denoted o. A return code *invalid_handle* indicates that the handle references no object, or if the referenced object does not exist at or below L_S.

Operations cannot assume that a handle presented as an argument is legitimate, or that a user who presents a handle is authorized to access the referenced object. Arguments come from untrusted user code, which may generate them randomly and maliciously, or may hide (e.g., as a String) a low handle to an object that has been upgraded. In the operations below,

---

[6] Handles are values that may be passed to application code, and one must assume that users have access to all information in the stored handle, such as (in some implementations) the fact that *d123* is an instance not of class *Department*, but of the subclass *Covert_Ops_Dept*.

the Boolean function *effectively_exists*(*h*) returns *true* iff *↑h* denotes an object *o* in the current database, and *o.exists* is dominated by L_S. It may be useful to consider this as an additional model function.

4.1. Accessing Elements: Retrieve, Modify

Attribute values are retrieved or modified only by the model's operations on elements.

Retrieve o.A : Traverses the association to return the element's value; also returns the element's level.

Interface:

Inputs: h: Handle, A: Attribute
/* The object referenced by *h* must have an attribute *A* in its class definition. The output value's type is determined by *A* */
Outputs: val: Value, L: Level

Semantics:

If *↑h* does not effectively exist, return *null* and the return code *invalid_handle.*
If there is no association from *o.A*, or that association is not dominated by L_S,
return *val* = *null* and the element's level.
Otherwise return the value and level for *o.A*.

Modify o.A = newValue: The value of element *o.A* is changed.

Interface:

Inputs: h: Handle, A: Attribute, newValue: Value
Outputs: none (except return codes)

Semantics:

If *o*= *↑h* does not effectively exist then return(code = *invalid_handle*).
If *level*(*o.A*) is not L_S then return (code = *wrong_level)*
assign *newValue* to *o.A*

Example:

Consider *p78.Address* in Figure 3.1. A confidential or secret request will read the confidential information that *level*(*p78.Address*) = "ts" and return *wrong_level*, thereby avoiding both polyinstantiation and the familiar covert channel. A top secret modify will succeed, while Unclassified will return *invalid_handle.*

4.2. Object Operations: Create, Delete, Test_Identity

Create: Create a new object

Interface:

Inputs: c: handle of a class
Outputs: h: a handle for the newly-created object o.

Semantics:

Produce a new object as an instance of the indicated class. Initialize every element's level to L_S. Assign *true* to *o.exist*. Return the object's handle.

Delete is a difficult operation with multilevel objects, because an object's existence spans levels. One form of deletion is presented here. Section 4.4 explores the possibility of greater flexibility.

Delete_up: Deletes the object's information at L_S or higher levels.

Interface:

Inputs: h: handle for an object (denoted o)
Outputs: none (except return codes)

Semantics:

If *h* does not effectively exist, do nothing and return. Otherwise, destroy all associations at L_S or higher from attributes of *o* (i.e., set these attribute values to *null*). If *exist* is at L_S, then *o* no longer effectively exists at L_S, the object is considered deleted, and *h* will no longer be a valid handle.

To the operations above, one might add an unprivileged operation *Test_Identity*, to test whether two handles reference the same object.

4.3 Operations that Manipulate an Element's Level

These operations alter the level for an element *o.A* (and hence for its association). When unprivileged, they both must run at level(*o.exist*). (At the end of the section, we discuss relaxing this restriction.) The operations are defined in terms of a hypothetical assignment method *set_level*(*h*, *A*, *new_level*) that also runs only at *level*(*o.exist*).

ReduceLevel o.A: Effectively resets the level of an element *o.A*, to its lowest value, discarding an association provided at a higher level.

Interface:

Inputs: h: Handle, A: Attribute
Outputs: none (except return code)

Semantics:

If (*h* does not denote an object *o* such that *level*(*o.exist*) = L_S) or
(*A* is not an attribute of *o*) then return //with an appropriate error code
Assign *o.A* = *null*; // destroy existing value, to avoid possible information flow
Set _level(*h*, *A*, L_S)

Regrade o.A: Changes the level of an element without losing its association. Any regrade except an upgrade issued at *level*(*o.exist*) requires privilege.

Interface:

Inputs: h: Handle, A: Attribute, newLevel: Level
Outputs: none (except return code)

Semantics:

If *h* does not denote an object for which *level*(*o.exist*) = L_S.
or *A* is not an attribute of *o* or
(*newLevel* does not dominate *level*(*o.A*) and request is not privileged)
then *error_return*
Otherwise *Set_Level*(*h*, *A*, *newlevel*)

We now sketch two ways to gain more flexibility. One approach is to require that *o.A* be classified to match its contents, e.g., it would be Secret that *level*(*o.A*) is Secret, TS<A> that it TS<A>, etc. This permits tighter protection of levels, and allows unprivileged upgrade by users who are above *level*(*o.exist*) but dominated by *level*(*o.A*). A user who is refused access to *level*(*o.A*) must assume that the level has been upgraded to a higher level. This does not appear to be a channel, when upgrades are initiated from below. Since these two approaches are so similar, it might be feasible to provide a single implementation that could be installed according to either convention.

A further generalization would be to make levels into ordinary labeled entities. However, that would necessitate storing an additional label; in both the previous cases the sensitivity of *level*(*o.A*) was determined by ordinary elements' labels (respectively *o.exist* or *o.A*).

4.4 Alternatives to the Delete Operation

To compare sv-UFOS with a DBMS that stores single-level objects, observe that an application object (e.g., *aShip*) that contains data at multiple levels would be represented by multiple DBMS objects; to delete it, the user must issue a *delete* command for each relevant

level, possibly from separate logins. Deleting an entire object is thus significantly easier in sv-UFOS. It is of little help however, in deleting selected portions of an object. This section investigates alternatives that cater to partial deletion. The tentative conclusion is negative—the generalization pays much of the implementation price of polyinstantiation, while providing much less expressive power.

We consider the alternative operation, *delete_at_level*, which sets elements at L_S to *null* without changing other levels. When L_S = *level(o.exist)* there is no problem. When L_S = *level(o.exist)*, *exist* must somehow be *rescued* and stored at the higher levels. It may also be desirable to rescue other elements of an object about to be deleted (e.g., *aShip.Captain*). Call the minimal level(s) above L_S at which the object has associations the *rescue_level(s)*.

We treat these difficulties as manifestations of a more general issue that also arises in nonsecure databases—the conflict between user autonomy and data sharing. One group (e.g., low) may provide data to another (e.g., high), but retain the right to update or delete information without permission from the consumers. The issue has been explored in conjunction with referential integrity [Morg90, Qian94], but can arise even among elements of a single object. The solution does not lie in changing the semantics of basic operations; rather, the consumer must create a private copy of required information, possibly when first reading it, possibly at the last moment using a trigger on *modify* or *delete*, or possibly by versioning.

Delete_at_Level is an operation of this sort. When there is a unique rescue level, the operation semantics appear straightforward. (The number of rescue levels is known at *level(o.exist)*). All rescued associations are regraded to the rescue level; this rescued information includes *exist* = *true* and (if specified in application-defined triggers) useful application elements that were at L_S. For this case, we can relax a restriction in [Morg90] and allow the rescuer to modify these associations at their new level.[7]

The case where there are incomparable rescue levels appears to require *weak polyinstantiation*, in which the *same* assertion is made at multiple security levels (in contrast to normal polyinstantiation in which assertions made at different levels may differ). Consider an object *aShip* which exists at Unclassified, but has additional attributes at two incomparable classified levels <A> and <B>, and at their upper bound <AB>. What is to be the result of a delete at the unclassified level? To be faithful to the outside world, the result should have a single object *aShip* that holds information in compartments <A>, <B>, and <AB>. References to *aShip* should continue to be usable by processes at these levels. Note, that one cannot speak unambiguously about the level of rescued elements.

Weak polyinstantiation causes operational difficulties. There appears to be no way to modify rescued attributes or to delete aShip from one compartment without causing a channel between compartments <A> and <B>.[8] We believe that polyinstantiation (of associations) provides a more flexible and cleaner way of separating actions at different levels [Rose94].

---

[7] Since it does not provide weak polyinstantiation of existence, the model in [Morg90] appears to require creation of a separate object for each rescue level.

[8] It may be possible to confine weak polyinstantiation to *exist*. To do so, one must relax the model's constraint that *aShip.Captain* must dominate *aShip.exist*; permitting this dubious state does not appear

5. DESIGN GOALS AND IMPLEMENTATION ISSUES

5.1. Design Goals

Our design process may have been unusual in the attention given to the goals of simplicity and robustness. These goals led to an effort to minimize restrictions within the kernel's states and operations. We therefore proposed that the model be specified separately from application programmer aids and integrity-promoting guidelines.

Model restrictions impose burdens on the DBMS's *specifiers* (to document and explain them), on DBMS *implementers* (to enforce them), on *security evaluators* (to evaluate the enforcement), on DBMS *maintainers* (if restrictions are altered), and especially on *users* (to use them or work around them).

To make the model simpler, we minimize the number of top-level concepts and avoid explicit restrictions. Existence is treated as a specialization of Element. Protection of existence, values, and object identifiers is built into operation semantics but does not require additional labeling. Moreover, we impose just one constraint on label assignments—that each element dominates *exist* (and even this can be relaxed). Other models vary in these respects. For example, due to use of object names, [Gajn88] uses seven inequalities on levels. Several models make difficult objects read-only. Metadata management is another area where we believe many inequalities should be relegated to guidelines outside the TCB.

Flexibility and robustness were also important design goals. Once the model imposes a restriction, it becomes difficult to relax because it may be assumed in code throughout the DBMS. Furthermore, so few applications have been built over MLS DBMSs that we cannot confidently declare any restriction harmless. Arguments that there is no utility to a particular database state often tacitly make assumptions that ought to be debated, for instance that states reachable only through privileged operations should be illegal.

A more conservative and robust course is to specify a general model, determine an implementation approach, and then accept only those restrictions that *greatly* simplify the implementation. Restrictions motivated by needs of the formal model rather than of users deserve great suspicion.

5.2. Implementation in a Client-Server Environment

As a model that labels individual elements, Sv-UFOS's data structures do not present new challenges. Conventional approaches include storing each object as a single multilevel record, or splitting it into multiple single level records.

OODBMSs do raise one important kind of design issue: How to pass information between the database (typically on a server) and the application (typically on a client). In relational systems, requests affect sets of tuples, which are then passed sequentially to or from the client. The client-side software is relatively simple. In OODBMSs the passed information

---

harmful. The rescue can then leave *aShip.Captain* unchanged (i.e., Unclassified). The operations' semantics still allow ordinary elements to be read only at levels that dominate *exist*, and will prevent updates.

is often much closer to the representation of data in storage, objects or pages. OODBMS client software maps from this direct representation and, upon request from a user program, directly retrieves or updates the desired elements. Eventually the object or page is written back to permanent storage.

The above scheme gives very good performance on OODBMS workloads, but raises serious security issues. In particular, if the client runs on a single-level or low assurance machine, any information passed to the client must be considered revealed, and any multilevel object or page returned from the client must be considered corrupted. We are currently investigating the degree to which various DBMSs' information-passing mechanisms can be protected from these vulnerabilities.

6. CONCLUSIONS

Sv UFOS is an object-oriented data model that provides fine-grained protection of data by protecting a single kind of security entity, namely elements. It contains many details that may be of use in future models. Existence information can use most operations defined on ordinary attributes. Element levels can be changed by unprivileged operations, though not as flexibly as we would like. Another contribution is that the model specifies operations in detail. Working out the details led to many improvements whose necessity was not obvious without operation definitions, especially for deletions and regrading.

We discussed the role of an MLS object model in an MLS DBMS, to distinguish the layer at which features should be specified. This practice helped in reducing the size of the TCB. The motivation for many design decisions was discussed. We devoted considerable attention to keeping the model's specifications consistent with priorities and practice in the OODBMS industry, especially compatibility with existing languages and good performance. We did not rely on encapsulation that violates C++ rules and is difficult to assure, did not change the language semantics to make integrity checks mandatory, and treated methods as procedures rather than security events.

We conclude that simultaneous avoidance of channels and polyinstantiation is feasible, but leads to users having considerably less flexibility (especially for Delete and Upgrade) than in the polyinstantiated UFOS model.

REFERENCES

[Bert94] Bertino, E., Mancini, L., Jajodia, S., "Collecting Garbage in Multilevel Secure Object Stores," *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. Oakland, CA, 1994.

[Bisk88] Biskup, J., and H. Brueggermann, "The Personal Model of Data: Towards a Privacy-Oriented Information System," *Computers & Security*, Dec. 1988.

[Catt91] Cattell, S., *Object Data Management*, Addison-Wesley, Reading MA, 1991.

[Gajn88] Gajnak, G., "Some Results from the Entity/Relationship Multilevel Secure DBMS Project," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pp. 66-71, April 1988.

[Jajo90] Jajodia, S., and B. Kogan, "Integrating an Object-oriented Data Model With Multilevel Security," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1990.

[Keef89] Keefe, T., W. Tsai, and B. Thuraisingham, "SODA - A Secure Object-oriented Database System," *Computers and Security*, Vol. 8, No. 6, 1989.

[Lunt94] Lunt, T. and P. Boucher, "The SeaView Prototype: Project Summary," *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD, Oct. 1994.

[Mark91] Markowitz, V., "Safe Referential Integrity Structures in Relational Databases," *Proceedings of the 17th International Conference on Very Large Databases*, Barcelona, Spain, 1991. (An extended version has been submitted to Information Systems).

[Mill92] Millen, J., and T. Lunt, "Security for Object-Oriented Database Systems," *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1992.

[Morg90] Morgenstern, M., "A Security Model for Multilevel Objects with Bidirectional Relationships," *Proceedings of the 4th IFIP Working Conference in Database Security*, Halifax, England, 1990.

[Oliv94] Olivier, M., and S. von Solms, "A Taxonomy for Secure Object-Oriented Databases," *ACM Transactions on Database Systems*, Vol. 19, No. 1, March 1994.

[Qian94] Qian, X., "Inference Channel-Free Integrity Constraints in Multilevel Relational Databases," *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. Oakland, CA, 1994.

[Rose93] Rosenthal, A. and W. Herndon, "Granularity of Data Protection for MLS Applications and DBMSs," *Proceedings of the 7th IFIP Working Conference on Database Security*, Huntsville, AL, August 1993.

[Rose94] Rosenthal, A., J. Williams, W. Herndon, and B. Thuraisingham, "Multilevel Security for Object-Oriented Database Management Systems," MITRE Technical Report (publication pending), The MITRE Corporation, Bedford, MA, 1994.

[Smit89] Smith, G., *The Modeling and Representation of Security Semantics for Database Applications*, Ph.D. Thesis, George Mason University, 1989.

[Thur91] Thuraisingham, B., "Multilevel Secure Object-oriented Data Model – Issues on Noncomposite Objects, Composite Objects, and Versioning," *Journal of Object-Oriented Programming*,, Vol. 4(7), Nov. 1991.

APPENDIX A. INTEGRITY ENFORCEMENT TRADEOFFS

We argue here that certain kinds of integrity enforcement should be performed only if requested by a data administrator who accepts their costs. We also discuss reasons why further research may be needed before MLS technology for integrity enforcement can be robust. Our arguments are based primarily on the nonsecurity requirements of the OODBMS market, rather than on covert channel issues. The reduction in the size of the Trusted Computing Base (TCB) is a bonus

We discuss referential integrity in detail and then mention some multilevel integrity constraints for which similar arguments apply.

A.1. Referential Integrity

A *referential integrity* constraint asserts that the database contain only valid pointers. That is, if some element's value is a nonnull handle (denoted h), then the database must contain an object o such that $o = \hat{}h$. One decision in UFOS that has proved controversial is that we allow handle-valued attributes for which referential integrity is not enforced. We advocate accepting reduced integrity primarily because we wish to conform to practice in nonsecure DBMSs where references are, by default, unchecked.[9]

Nonsecure DBMSs accept suboptimal integrity in order to obtain compatibility and performance. If all references were subject to integrity checks, several problems would arise:

- Existing programs would now crash whenever they performed updates that invalidated references. Such programs need not be incorrect—they may simply defer the reference check till the reference is used.

- When a value is assigned to *any* element of type *Handle*, one must check that the reference was valid (since the argument comes from untrusted user code). Thus, an assignment that could have been performed on a client might now require access to disks on a server. One might also need to update a object reference count, possibly requiring that an update go to the server.

- The human creator of an object o may insist on the right to delete it, even if another user holds a handle. Referential integrity can then be enforced only if the DBMS maintains a costly *inverse-traversal* structure that, for each o, identifies all elements whose values are handles of o. Every update to a handle-valued element causes an update to this structure.

We do believe that the data administrator ought to have the option of requesting enforcement on any reference-valued attribute. However, even in non-MLS systems, the

---

[9] C++ and Smalltalk offer no referential integrity checks, while OODBMS products often check only when explicitly requested. In relational systems, an attribute can be joined with a foreign key even if no foreign key constraint has been declared; these are references with neither integrity checks nor type checks.

Creating Abstract Discretionary Modification Policies with Reconfigurable Data Objects

Todd Gross
Dept. of Computer Sciences
The University of Texas at Austin*
tag@cs.utexas.edu

July 25, 1994

ABSTRACT

We desire a mechanism for defining security policies that is both flexible and easy to use. This allows us to manage dynamic changes to database schemas and security policies with the least amount of effort. Any such mechanism will therefore have to be (1) discretionary (for changing the security policy) and (2) defined over abstract data (for changing the database schema). Currently, the only security mechanisms with these features operate on *views*. Despite their abstractness, specifying a policy using views requires detailed knowledge of how the views are implemented. Furthermore, because of the view update problem, we need a special mechanism for enforcing modification policies over views, one that reveals the structure of the underlying data to any user that is allowed to modify the database.

This paper proposes a new data abstraction called the *reconfigurable data object*, with properties that simplify the creation of abstract security policies. In particular, it will allow us to define modification policies over our database while hiding implementation details from both the end user and, potentially, the person responsible for specifying and maintaining the overall database security policy.

*This research was not funded by any grant whatsoever.

1 Introduction

Abstraction is a fundamental concept in computer science: it simplifies the process of solving problems by eliminating details of the problem space that are not relevant to solving the problem. Maintaining the security of information stored in a database is a problem we are interested in solving, and we use abstraction to ignore details about data representation and database implementation in creating a set of mechanisms to enforce security. Abstraction is a particularly useful concept for this problem, because we can also use it to hide details about database implementation from the DBMS user, making the DBMS easier to use and increasing its security. This explains why views have been used by database administrators to restrict access to information stored in a database.

Views are, however, an insufficient abstraction for specifying *modification* policies—policies that determine when data may and may not be modified—because views in general cannot be updated directly. Furthermore, views do not hide their own implementation details, and therefore policies over them can be just as complex and "implementation dependent" as policies over base relations. What we desire is a data abstraction that hides its own implementation details as well as those of the underlying data. Security policies defined over such an abstraction would be easier to specify (because there are fewer details to remember) and more secure (because less is known about how the database is implemented).

This paper is about just such an abstraction, which we call the *reconfigurable data object* (or RDO for short). With this abstraction, we can create security policies that are discretionary and schema-independent.

Section 2 defines reconfigurable data objects and provides a few examples. Section 3 shows how we can specify modification policies over RDOs. Section 4 defines algorithms for enforcing these security policies, and demonstrates how they work. Section 5 compares view-based and RDO-based modification policies through an example (the former differing radically from standard view-based policies). Finally, Section 6 offers some concluding remarks.

2 RDOs and Related Concepts

Before we can define what an RDO is, we need to explicate the concept of a *function* over a database—in our case, a relational database. Many existing DBMSs allow the creation of functions that consist of a query in their own DML containing one or more *parameters* to the function. When the function is called, the parameters are bound to the values passed as arguments to the function, and the query is evaluated. For example, consider the following EXCESS function [CDV87, p 24]:

```
define Employee function youngerKids(maxAge: int4) returns int4
(
    retrieve (count(C from C in this.kids where C.age < maxAge))
)
```

The function youngerKids has one parameter (maxAge), and its body is an EXCESS query that uses maxAge in the **where** clause of the query. The value returned by the function call youngerKids(10) is the value returned by the body of the function with maxAge bound to the value 10—in other words, the number of children of all people in the Employee relation that are less than 10 years old. Definition 1 formalizes this concept.

Definition 1: A *database function* f is a function of one or more parameters whose body is a SQL query with the following properties:

- All tuple variables are explicitly named
- All parameters of f appear at least once in the WHERE clause □

Figure 2.1(a) defines a database function salary that satisfies Definition 1: it has a single parameter emp that is used in the WHERE clause of the function, and the body of the function is a SQL query with one tuple variable named E. When the function is called with an employee ID number as an argument, the function will return the salary of that employee. But in order for this to work, the function query must be evaluated against a database with a schema that supports the relations and attributes used in salary. Without such a database, we cannot evaluate a call of function salary. Definition 2 formalizes this concept.

Definition 2: Let f be a database function, and $\mathcal{D}$ be a relational database. $\mathcal{D}$ *supports* f iff:

1. For every relation name R in the FROM clause of f, R is in the schema of $\mathcal{D}$
2. For every expression of the form $V.attr$, where V is a tuple variable over relation R in the body of f, *attr* is an attribute of R in the schema of $\mathcal{D}$ □

The database in Figure 2.1(b) supports function salary in Figure 2.1(a): it contains a relation named *Employee*, which contains attributes wage (from the SELECT clause) and eid (from the WHERE clause). The relation also has attributes name and dept, which are not used by the database function and therefore do not affect the satisfaction of Definition 2.

With a formal definition of database functions, and the support of such a function by a database, we can now give a formal definition of the *value* of a call to a database function. This definition is given below, with examples of its use given in Figure 2.1(c).

Definition 3: Let f be a database function, $\mathcal{D}$ be a database that supports f, and $\vec{a}$ be a list of values. The value returned by a call of f with argument list $\vec{a}$ evaluated against $\mathcal{D}$ (denoted $f(\vec{a}, \mathcal{D})$) is the value returned by evaluating the body of f, with the parameters of f bound to their respective values in $\vec{a}$, against the corresponding tuples of $\mathcal{D}$. □

Figure 2.1(c) shows the value returned by two different calls to our database function salary when evaluated against the database in Figure 2.1(b). For the first call, there is only one assignment of tuples to our tuple variable that satisfies the WHERE clause of salary—namely, assigning E the fourth tuple of

Employee—therefore the function returns a singleton set. For the second call, there is no such assignment (as there is no tuple in *Employee* with an `eid` attribute of 21250), therefore the call returns the empty set.

```
salary(emp)
{
    SELECT  E.wage
    FROM    Employee E
    WHERE   E.eid = emp
}
```

(a)

Employee

eid	name	dept	wage
100	Aaron	1	55000
101	Baker	1	35000
102	Chase	1	37500
200	Davis	2	21250

(b)

```
salary(200)          =  {21250}
salary(salary(200))  =  {}
```

(c)

Figure 2.1: The creation and use of database functions: (a) a database function definition, (b) a sample database containing the relation used by our database function, (c) two sample database function calls and their values given our sample database.

Having formalized the concept and attributes of database functions, we now define the basic abstraction of our security model: the reconfigurable data object.

Definition 4: Let f be a database function and $\vec{a}$ be a list of values such that for some database $\mathcal{D}$ that supports f, $f(\vec{a}, \mathcal{D})$ is nonempty. Then the pair $\langle f, \vec{a} \rangle$ is a *reconfigurable data object* (or *RDO*). □

An RDO is a syntactic construct consisting of a database function name and a proper list of arguments to the function. For example, $\langle \texttt{salary}, \langle 200 \rangle \rangle$ is the RDO that corresponds to the first function call in Figure 2.1(c). Do not confuse the RDO $\langle f, \vec{a} \rangle$ with the function call $f(\vec{a}, \mathcal{D})$, RDOs represent abstract values that are independent of the structure and contents of the database, and independent of the function bound to f. This not only reduces the amount of information a user needs to know to obtain a value from the database, it also allows great flexibility in reconfiguring the structure of the database. Furthermore, and for our purposes most importantly, this abstraction hides both the contents of the database and the algorithm used to evaluate the RDO from the user, making inference of unauthorized information that much more difficult.

In the next section, we show how to define modification policies with RDOs as the unit of protection, and what such policies mean.

3 Creating Policies over RDOs

We have created a syntax for specifying security policies over RDOs, the syntax was designed to be easy to use and to support a wide range of security policies. Ease of use again means abstraction—there are no

implementation details in the policy specification—but it also means keeping track of as few details about the policy itself as necessary. Supporting a wide range of policies means that our policies are discretionary, as mandatory policies place tight restrictions on access and modification based on information flow.

3.1 Syntax

A policy, in our syntax, is simply a list of *statutes*, each statute placing specific restrictions over a set of RDOs based on properties of the RDO. Since we are interested here in modification policies, we will only describe the syntax of statutes on modification. Such statutes can take one of two formats:

Format 1: ALLOW MODIFY $f(\vec{p})$ WHERE $pr(\vec{p})$
Format 2: DISALLOW MODIFY $f(\vec{p})$ WHERE $pr(\vec{p})$

where f is a database function, $\vec{p}$ is a list of parameter names corresponding to the parameters of f, and $pr(\vec{p})$ is a boolean predicate over the parameters named in $\vec{p}$. Informally, each statute specifies a list of RDOs whose value the DBMS user is allowed (or not allowed) to modify. This includes every RDO of the form $\langle f, \vec{a} \rangle$, where f is the database function mentioned in the statute and $\vec{a}$ is a list of arguments to f that cause $pr(\vec{p})$ to evaluate to TRUE when parameter list $\vec{p}$ is bound to $\vec{a}$. For example, the statute

```
ALLOW MODIFY salary(emp) WHERE emp = 200
```

allows a user to change the value associated with RDO $\langle \texttt{salary}, \langle 200 \rangle \rangle$, which corresponds to the salary of the employee with ID number 200. Note that this statute makes no reference to any property of the person performing the query (in particular, no reference to a user clearance level), and therefore applies to any DBMS user. If we want the semantics of a statute to depend on who is performing the modification, we need some extra syntax. If, for example, we let $EID be a token that evaluates to the employee ID number of the current DBMS user, the statute

```
DISALLOW MODIFY salary(e) WHERE e = $EID
```

prevents the user from modifying the value of RDO $\langle \texttt{salary}, \langle \$\texttt{EID} \rangle \rangle$, which corresponds to the user's own salary.

The previous two example statutes have been quite simple: the predicate compares a single parameter to a constant value. More complex policies will be not merely value-based but *content-based*, dependent on the current contents of the database. To ensure that our policies remain independent of the database schema (like the RDOs over which our policies are defined), we require that all accesses to the database be through database function calls. For example, in the statute

```
ALLOW MODIFY salary(emp) WHERE salary($EID) > salary(emp)
```

the function salary is called twice in the predicate. This statute allows the user to modify the salary of any employee whose salary is smaller than their own. The statutes assume the existence of a current database that supports all database functions used by a policy. Thus, if we make the database in Figure 2.1(b) the current database, the preceding statute allows employee Aaron to modify every employee's salary but her own.

3.2 Semantics

In the previous section, we defined the semantics of statutes in terms of the set of RDOs that satisfy the predicate of the statute. Taking our first statute as an example, the only value we can assign to emp to satisfy the predicate emp = 200 is 200—therefore the only RDO we allow the user to modify under this statute is $\langle \texttt{salary}, \langle 200 \rangle \rangle$. But the abstractness of our RDOs presents a semantic issue we don't face with more concrete data objects like relations and tuples: their contents can *overlap*. This means that when we modify the value of one RDO we may also be modifying the value of other RDOs at the same time. For example, consider the database function pay defined in Figure 3.1(a) below:

```
pay(emp)
{
    SELECT  E.wage + D.bonus
    FROM    Employee E, Department D
    WHERE   E.eid = emp AND D.did = E.dept
}
```

(a)

Department

did	dname	bonus
1	Admin	1000
2	Sales	100
3	R&D	0

(b)

Figure 3.1: Example of potential overlap of RDOs (a) new database function (b) extension to the database to support the new function

If we wanted to change the value of salary($emp, \mathcal{D}$) for a given *emp* and $\mathcal{D}$, we must change the value of the wage field of the corresponding tuple in *Employee* in $\mathcal{D}$. For example, if $\mathcal{D}$ is our sample database, then changing the value of wage in the fourth tuple of *Employee* will change the value of $\langle \texttt{salary}, \langle 200 \rangle \rangle$ accordingly. If we now add the relation *Department* from Figure 3.1(b) to $\mathcal{D}$, the same modification to the wage field changes the value of $\langle \texttt{pay}, \langle 200 \rangle \rangle$ as well as $\langle \texttt{salary}, \langle 200 \rangle \rangle$. Therefore the statute

```
ALLOW MODIFY salary(emp) WHERE emp = 200
```

must allow us to modify both of these RDOs, as one cannot change the value of $\langle \texttt{salary}, \langle 200 \rangle \rangle$ without also modifying the value of $\langle \texttt{pay}, \langle 200 \rangle \rangle$. This raises a semantic issue with respect to policies: if a modification m of the contents of a database $\mathcal{D}$ changes the value of two distinct RDOs $\langle f_1, \vec{a}_1 \rangle$ and $\langle f_2, \vec{a}_2 \rangle$, when should we allow m to take place: (a) when **both** RDOs satisfy an ALLOW MODIFY statute from the policy, or (b) when **at least one** RDO satisfies an ALLOW MODIFY statute of the policy? We opt for the latter interpretation, because it reduces the amount of information a policy writer needs to know. In particular, if a new database

function is added to the system, the policy writer need not worry about adding ALLOW MODIFY statutes over the new function just to maintain the current policy.

Another unusual semantic feature of RDOs is that one can change the contents of the database without changing the value of any RDO. In fact, it is possible to change the value of tuple fields used to calculate an RDO without changing the actual value calculated. For example, if we change the wage field of the last tuple in Figure 2.1(b) from 21250 to 21300 and the bonus field of the second tuple in Figure 3.1(b) from 100 to 50, the value pay(200, $\mathcal{D}$) is still 21350, even though both of these fields are used to calculate this value[1]. From our standpoint, any change to the database that doesn't change the value of any RDO doesn't count as a modification, and therefore will be allowed regardless of the current policy. To sum up, the set of RDOs that an ALLOW (DISALLOW) MODIFY statute will allow (not allow) to be modified is:

- RDOs whose database function matches the statute function and whose argument list satisfies the statute predicate
- any RDOs whose value changes as a result of modifying said RDOs (because of overlap)

Any RDOs whose value doesn't change as a result of a modification to the database have no effect on whether or not the modification is allowed.

As a policy is simply a list of statutes, the set of RDOs one can modify under a policy is based on the individual statutes of the policy. An RDO r may be modified under policy $\mathbf{P}$ iff

- There is at least one ALLOW MODIFY statute $s_A \in \mathbf{P}$ that covers r
- There are no DISALLOW MODIFY statutes $s_D \in \mathbf{P}$ that cover r

The syntax and semantics we've provided for specifying modification policies is flexible (because it's discretionary and value-based) and abstract (because all references to data are made through database function calls). But how would one enforce such a security policy? This is the subject of the next section.

4 Enforcing Our Policies

In existing discretionary security policies, the units of protection are sufficiently concrete that enforcing security is straightforward. System R for example protects tables and views, and one can simply maintain a list for each user of the tables and views they are permitted to access. Because of the abstractness of RDOs, one cannot simply maintain a list of which database objects the user is allowed to operate on: policies can depend on the contents of the database and database functions may be rewritten to keep up with changes to database schemas. Essentially, we need to dynamically determine which RDOs have changed value when a change is made to the database. For example, a change in the bonus field of the first tuple of *Department* will change the value of three RDOs: ⟨pay, ⟨100⟩⟩, ⟨pay, ⟨101⟩⟩, and ⟨pay, ⟨102⟩⟩.

[1] This assumes that both modifications occur within a single transaction.

This task is quite complex in general, so we have made several simplifying assumptions, which we will explain as we go along. The following subsection gives a brief outline of our approach to solving this task, the remaining subsections briefly explain the algorithms for achieving this solution. There will be a running example of a policy and its enforcement to demonstrate this process.

4.1 Outline of Approach

Given a set of database functions $\mathcal{F}$, a database $\mathcal{D}$ that supports $\mathcal{F}$, a modification policy $\mathbf{P}$ whose statutes use functions from $\mathcal{F}$, and a modification m of the contents of $\mathcal{D}$, our task is to determine whether m is permitted under $\mathbf{P}$. We divide this task as follows:

1. For each database function f that is covered by a statute in $\mathbf{P}$, determine whether f belongs to a subclass called *conjunctive locator functions*. If not, all statutes in $\mathbf{P}$ that cover f are ignored.
2. For each function f_l that belongs to this subclass, construct an *internal table* of all RDOs over f_l that have nonempty values when evaluated against $\mathcal{D}$. Store the value of the RDO with its identifying information.
3. Perform modification m on $\mathcal{D}$.
4. Re-evaluate every RDO r in the internal tables and compare the new value with the stored value. If the two values are different, add r to a set R.
5. Compare every RDO $r \in R$ to every statute $s \in \mathbf{P}$ to determine whether s covers r. If there is any `DISALLOW MODIFY` statute in $\mathbf{P}$ that covers any $r \in R$, undo the modification. Otherwise, if there is no `ALLOW MODIFY` statute in $\mathbf{P}$ that covers any $r \in R$, undo the modification. Otherwise let the modification stand.

We now consider each of these steps in turn, with the following running example to demonstrate our approach: let $\mathcal{F}$ contain the two database functions previously defined (`salary` and `pay`), $\mathcal{D}$ contain the two relations previously defined (*Employee* and *Department*), and $\mathbf{P}$ be the following policy:

```
ALLOW MODIFY salary(emp) WHERE emp = $EID
DISALLOW MODIFY pay(emp) WHERE salary(emp) < 30000
```

4.2 Before the Modification

Before we are ready to enforce our policy, we must ensure that we are in the proper initial state. Basically, we must (a) record the existence of every RDO whose value could be affected by the modification and (b) record the current value of these RDOs so that we can compare this value to the value after the modification. This process is difficult to generalize, therefore we have restricted our attention to a subclass of database functions called *conjunctive locator functions*, for which we have defined algorithms that are provably correct with respect to our intended semantics. This subclass is defined as follows:

Definition 5: A database function f is a *conjunctive locator function* iff:

- For any database $\mathcal{D}$ that supports f and RDO $\langle f, \vec{a} \rangle$, $f(\vec{a}, \mathcal{D})$ evaluates to either the empty set or a singleton set

- For any list of tuples $\vec{t}$ that may be assigned to the tuple variables of f, there is no more than one RDO $\langle f, \vec{a} \rangle$ such that the `WHERE` clause of f is satisfied when the parameters of f are bound to $\vec{a}$ and the tuple variables of f are bound to $\vec{t}$
- The `WHERE` clause of f is a conjunction of equality tests among (a) function parameters, (b) tuple variable attributes, and (c) literal values □

Both of the database functions in our running example are conjunctive locator functions. Functions with these properties simplify the creation of *internal tables*, which are used to record the current value of all RDOs with nonempty values.

Definition 6: Given a conjunctive locator function f and database $\mathcal{D}$ that supports f, the *internal table* for f over $\mathcal{D}$ (denoted $T(f, \mathcal{D})$) is a relation with the following properties:

- The relation schema contains one attribute for each parameter of f and one value attribute.
- For every RDO $\langle f, \vec{a} \rangle$ such that $f(\vec{a}, \mathcal{D})$ is nonempty, there is a tuple t in $T(f, \mathcal{D})$ such that (a) the attribute for a given parameter of f contains the value of the corresponding argument in $\vec{a}$ and (b) the value attribute contains the value $f(\vec{a}, \mathcal{D})$. □

Figure 4.1 shows the internal tables for our two database functions `salary` and `pay`. Notice that there is one tuple in each of these tables for every tuple in *Employee*. We give an algorithm for calculating the contents of an internal table below: the restriction of our method to conjunctive locator functions is primarily to simplify the creation of these internal tables.

Algorithm 1 (creating internal tables):
For every legal assignment of tuples in $\mathcal{D}$ to tuple variables in f:

1. Substitute the values of the fields of the corresponding tuples for the tuple attribute expressions in the body of f
2. If there is an assignment of values to the parameters of f such that the `WHERE` clause evaluates to `TRUE`, add a tuple to $T(f, \mathcal{D})$ assigning each parameter field the value bound to that parameter and assigning the value field the value of the expression in the `SELECT` clause of f

To see how this algorithm works, let f be `salary` and assign tuple variable `E` the first tuple of *Employee*. Then `E.eid` is assigned the value 100, and we set parameter `emp` to 100 to make the `WHERE` clause evaluate to `TRUE`. Since we were able to satisfy the clause, we add a tuple to $T(\texttt{salary}, \mathcal{D})$ with an `emp` value of 100 and the value attribute set to `E.wage` or 55000.

$T(\texttt{salary}, \mathcal{D})$

emp	*value*
100	55000
101	35000
102	37500
200	21250

$T(\texttt{pay}, \mathcal{D})$

emp	*value*
100	56000
101	36000
102	38500
200	21350

Figure 4.1: The internal tables generated by using Algorithm 1 against database $\mathcal{D}$

4.3 Making the Modification

Once the internal tables have been properly set up, we can modify the database. Note that our modification policy says nothing about **how** to modify the database—all we need to know is which fields were modified and what their new values are. Views, on the other hand, provide a fixed interface to the underlying database that may not map uniquely to the database. For our running example, we will change the value of the `bonus` field in the second tuple of *Department* from 100 to 500.

Having made the modification, we need to determine which of our RDOs has changed value. We do this by taking each tuple t of each internal table $T(f, \mathcal{D})$, extracting the argument list $\vec{a}$ from t, calculating the value $f(\vec{a}, \mathcal{D})$, and comparing this value with the stored value attribute in t. If they are **not** equal, we add the RDO $\langle f, \vec{a} \rangle$ to set R—which is initially the empty set. We currently have two internal tables with four tuples each.

Starting with the first tuple of $T(\texttt{salary}, \mathcal{D})$, we extract the argument list from the tuple. The argument list for this internal table is simply $\langle \texttt{emp} \rangle$, and therefore $\langle 100 \rangle$ for the first tuple. We then calculate $\texttt{salary}(100, \mathcal{D})$, where $\mathcal{D}$ is the new modified database, and get 55000[2], which is the same as before. We therefore do not modify R. In fact, no RDO over `salary` will change value, as no modification was made to *Employee*, which is the only relation used by the function. We therefore skip to the first tuple in the `pay` internal table.

We evaluate $\texttt{pay}(100, \mathcal{D})$ by assigning tuple variable `E` the first tuple in *Employee* and assigning `D` the first tuple in *Department*, because that assignment satisfies the `WHERE` clause. Since we modified the second tuple of `Department`, not the first, the return value is the same as before, namely 56000. The only RDO that changed value is $\langle \texttt{pay}, \langle 200 \rangle \rangle$, represented by the last tuple in the `pay` internal table. We therefore add this RDO to our (previously empty) set R and quit checking values. At this point, we know of every RDO over our database functions that has changed value with respect to the current database.

We should note that this process of calculating which RDOs have changed value is very expensive, both in terms of time and space. We have to create an internal table for every function we are interested in, add a tuple to that table for every RDO that has a nonempty value using our current database, and that's before we even start enforcing our policy. Once a modification is made, we need to recalculate the value of every RDO, even though most will not change value. And each of these evaluations corresponds to a separate query over our database, with a separate scan of all the relevant relations. We intend to develop a more efficient algorithm for discovering which RDOs have been modified as our research progresses.

4.4 Checking its Correctness

Once we have the set of RDOs whose value has been modified, we can compare them against the statutes of our policy to see whether they allow (or disallow) the RDO to be modified. As with other positive/negative

[2]To be precise, we get {55000}, as functions return sets as values. But because conjunctive locator functions will never return a set of more than one element, we coerce the set into its single element.

permission mechanisms[3] (e.g. GRANT/REVOKE in System R [ABC+76] and ALLOW/DISALLOW in RRDS [GO90]), we allow modification when at least one positive permission (for us, an `ALLOW MODIFY` statute) covers the RDO and no negative permissions (`DISALLOW MODIFY` statutes) cover the RDO[4]. But in our case RDOs can overlap, therefore we have to extend this a bit: a modification is permitted if **any** of the RDOs in our set R are covered by an `ALLOW MODIFY` statute in our policy **P** and **none** of these RDOs are covered by a `DISALLOW MODIFY` statute in **P**.

In our running example, the set of changed RDOs R has a single element, which is $\langle \text{pay}, \langle 200 \rangle \rangle$. We see that the first statute in **P** does not cover this RDO, regardless of who made the modification, because this statute only applies to RDOs over function `salary`. The second statute is over the correct function, therefore we test whether the predicate holds over this RDO—that is, whether `salary(200) < 30000`. Since `salary(200)` = 21250 when evaluated against our current database, the predicate holds. In sum then, no `ALLOW MODIFY` statute holds and one `DISALLOW MODIFY` statute holds. Therefore, the modification is not allowed and must be undone.

Since there were no changes to the database, the internal tables as they stand are correct, so we can go directly to step 3 of our enforcement algorithm (from Section 4.1) and perform the next modification. Had the modification been allowed (e.g. a modification of the `wage` field of the first *Employee* tuple by employee Aaron) we would have to replace the value attribute for all RDOs in R to reflect the modification (in this case, $\langle \texttt{salary}, \langle 100 \rangle \rangle$ and $\langle \text{pay}, \langle 100 \rangle \rangle$). Then we could proceed to step 3 as before.

5 A Comparison of Views and RDOs

We are now in a position to compare the expressive power of relational views and RDOs as a basis for defining security policies. In this section, we will define an abstract policy (based on our existing relations *Employee* and *Department*), represent the abstract policy using both views and RDOs as basic data objects, and discuss how one would enforce such a policy as defined for both of our data abstractions.

5.1 Policy Representation

First, let us define an abstract policy (where by *abstract* we mean independent of the underlying data representation). Let our abstract policy $\mathcal{P}$ be defined as follows:

> $\mathcal{P}$: An employee may modify the pay of all employees who work in their department, but no others

In order to implement this policy, we will need abstract definitions of the concepts "pay" and "department" of a given employee. For RDOs, we use the database function pay defined in Figure 3.1 to define the concept of "pay," the equivalent definition using views is given in Figure 5.1 below.

[3] It should be noted that *profiles* as described in [OvS92] take a different approach: all positive permissions (which they term *necessary conditions*) must be satisfied.

[4] A statute *covers* an RDO when binding the parameters of the database function (which immediately follows the word `MODIFY` in the statute) to the arguments in the RDO causes the predicate of the statute to evaluate to TRUE.

```
CREATE VIEW PayView(id, pay) AS
SELECT  E.eid, E.wage + D.bonus
FROM    Employee E, Department D
WHERE   E.dept = D.did
```

Figure 5.1: A representation of "pay" using a view

Note that we have two attributes in our view PayView: one to hold the pay amount and one to identify the employee whose pay it represents. We also need representations for the abstract concept of an employee's department, these are given (for views and RDOs) in Figure 5.2.

```
dept(emp)
{
    SELECT  E.dept
    FROM    Employee
    WHERE   E.eid = emp
}
```

(a)

```
CREATE VIEW DeptView AS
SELECT eid, dept
FROM Employee
```

(b)

Figure 5.2: Representations of an employee's department using (a) database functions and (b) views

Again, notice that the definition of DeptView includes a field to identify the relevant employee, whereas the database function dept takes this identifier as an argument which it uses to calculate the appropriate value. The difference between these approaches becomes clear when we use these definitions to represent our abstract policy $\mathcal{P}$, as shown in Figure 5.3.

$\mathcal{P}_{\text{View}}$:
```
ALLOW MODIFY P.pay
FROM   PayView P, DeptView D1, DeptView D2
WHERE  P.id = D1.eid  AND  D2.eid = $EID  AND  D1.dept = D2.dept
```

$\mathcal{P}_{\text{RDO}}$:
```
ALLOW MODIFY pay(emp) WHERE dept(emp) = dept($EID)
```

Figure 5.3: Representations of abstract policy $\mathcal{P}$ using views and RDOs

As you can see, $\mathcal{P}_{\text{RDO}}$ is not only simpler and shorter than $\mathcal{P}_{\text{View}}$, but more abstract as well. In the view policy, the attributes we created to store the employee ID number (id in PayView and eid in DeptView) were mentioned by name. The RDO policy, on the other hand, only mentions the value of an employee ID (through the database function(s) parameter emp) and not how that value is internally represented. The implementation independence of $\mathcal{P}_{\text{RDO}}$ makes it feasible to separate the duties of policy administration and database administration, a feature we will expand on in the concluding remarks.

5.2 Policy Enforcement

For a policy specification to be of any use, we must be able to enforce the policy as specified. In the case of abstract policy $\mathcal{P}$, this means that we must allow all modifications m by a user u to a database $\mathcal{D}$ that

is covered by $\mathcal{P}$ where m does not change the pay of any employee e in a different department than u. For example, we would allow u to modify the `bonus` field of the *Department* record corresponding to u but not for any other *Department* record.

Notice that we could not modify the `pay` attribute of `PayView` directly, because this attribute is the sum of a `wage` field and a `bonus` field, and there is more than one way to modify these two fields to obtain the new value of `pay`. Cases like this where a change to a view maps nonuniquely to changes in the base relations that populate the view is what we call the *view update problem*—note that this is a different problem than the one cited in §3.6.2 of [KS91], which concerns null key fields in base relation tuples created from a view tuple—and requires us to restrict modifications to base relations only (e.g., *Employee* and *Department* in the case of `PayView`). The algorithm for enforcing policy $\mathcal{P}_{\text{View}}$ against these changes is given in Figure 5.4 below:

1. Update `PayView` to reflect the changes to our base relations
2. Note all records in `PayView` whose pay attribute has changed
3. Assign each such record to tuple variable `P` from the WHERE clause of $\mathcal{P}_{\text{View}}$
4. For each tuple assigned to `P`, determine if there are two tuples in `DeptView` that can be assigned to tuple variables `D1` and `D2` such that the WHERE clause of $\mathcal{P}_{\text{View}}$ is satisfied
5. If there are any tuples bound to `P` that have no corresponding bindings to `D1` and `D2` that satisfy the WHERE clause, the modification must be undone. Otherwise, the modification stands.

Figure 5.4: Algorithm for enforcing $\mathcal{P}_{\text{View}}$

The algorithm for enforcing $\mathcal{P}_{\text{RDO}}$, on the other hand, is given in Figure 5.5. Note that the algorithm is fundamentally the same as in Figure 5.4 despite the surface differences: database function evaluation replaces the relational scan of $\mathcal{P}_{\text{View}}$, and variable binding replaces much of the equijoin computation.

1. Update the internal table for database function `pay` to reflect the changes to our base relations
2. Retrieve the `emp` attribute for each tuple of $T(\texttt{pay}, \mathcal{D})$ whose value attribute changed
3. Find all tuples in $T(\texttt{dept}, \mathcal{D})$ with the same value in their `emp` field as was retrieved
4. Compare the value attribute of all such tuples to `dept($EID)`. If any of our changed tuples from the `pay` internal table has no corresponding tuple in the `dept` internal table with this value as its value attribute, the modification must be undone. Otherwise, the modification stands.

Figure 5.5: Algorithm for enforcing $\mathcal{P}_{\text{RDO}}$

The algorithm for enforcing $\mathcal{P}_{\text{RDO}}$ is simpler than the algorithm for enforcing $\mathcal{P}_{\text{View}}$, because `dept($EID)` remains constant as the former algorithm is evaluated. Therefore, each `pay` table record whose value has changed will only require a single-level scan of the `dept` internal table, whereas we need a two-level scan of `DeptView` to retrieve tuples `D1` and `D2` to satisfy $\mathcal{P}_{\text{View}}$. This simplification, however, should not be seen as a significant advantage—as a smart optimizer could conceivably make a similar improvement to $\mathcal{P}_{\text{View}}$. The primary advantage of using RDOs/database functions to specify an abstract policy versus views is the implementation independence and simplicity of the former.

6 Conclusions

Nowadays, we expect software to be both powerful and easy to use. Word processors, for example, allow the user to use many different typefaces and sizes of text, with control over spacing, margins, alignment, and other aspects of the document. And yet these packages are simple to learn and use, because the user need not worry about how information concerning typefaces, font size, etc. is stored in their document. That is, the word processor provides an abstract interface (via a WYSIWYG display screen) that hides the internal representation of the text from the user, yet provides her powerful tools for creating documents. This abstract interface also allows the user to *import* documents created using other word processing packages, as the interface the user sees is independent of how the formatting and textual information is represented. For database security systems, however, this expectation is not easily met.

For these systems, the user of interest is the person who creates and maintains the security policy for a specific computing environment. This policy writer would like to create policies like $\mathcal{P}$ from the previous section—referring to high-level abstract concepts like "salary" and "department" without reference to implementation details like field and relation names. Mandatory security is defined directly over concrete relations, tuples, and fields; therefore, policies are implementation dependent. Furthermore, they are restricted to information flow based policies. Views do provide abstraction in that they hide the underlying data representation, but one must still know the schema for these views to write security policies using them, as we saw with $\mathcal{P}_{\text{View}}$. Furthermore, enforcing $\mathcal{P}_{\text{View}}$ requires functionality that existing view-based systems don't possess. DB2, for example, will not allow a user to modify the contents of a joined view like `PayView`, which is different than preventing the contents from changing value.

On the other hand, $\mathcal{P}_{\text{RDO}}$ only required that the policy writer knows how to call database functions `pay` and `dept`. The implementation independence of RDO-based policies allows us to separate the duties of policy management and database management. The database manager would manage the data and define database functions over them, the policy writer would create sets of statutes using these functions. This leaves the database manager free to change the database schema as the environment changes without worrying about how it affects security[5]. Note that this separation of powers is the same form espoused by Clark and Wilson in their seminal paper [CW87]. This work is also related to [Cab93], in that data in "physical" form must satisfy constraints that are written over "abstract" data—but in her case, the physical-to-abstract transformation must preserve the structure of the data (and functions like pay do not).

This advantage in flexibility and ease of use, however, comes with a heavy price tag. Enforcing a policy is highly time- and space-consuming. Policies are currently limited to a relatively small class of database functions[6], because other classes are even more costly to enforce. All database access is through database functions, therefore one loses the flexibility of SQL and may have to write hundreds of database functions, which would substantially increase the cost of enforcing policies. Finally, like views themselves, verification

[5] See [Sjø92] for an example of a database whose schema changed dramatically in a relatively short period of time.

[6] The class is however sufficient, in that one can use conjunctive locator functions to retrieve any attribute of any tuple of any relation with unique keys.

of code would be difficult, as the enforcement algorithm relies on much of the DBMS to function.

There is much work left to be done. We need to find more efficient algorithms for enforcing security policies. The class of database functions we can and should monitor needs to be investigated. There are many extensions we could make to the current policy syntax, several are worth pursuing. Finally, an implementation of these concepts needs to be fleshed out to test the effectiveness of this approach. It is clear that we have only laid the groundwork, and we have many questions to answer before this approach can be deemed practical.

References

[ABC+76] M M Astrahan, M W Blasgen, D D Chamberlain, K P Eswaren, J N Gray, P P Griffiths, W F King, R A Lorie, P R McJones, J W Mehl, G R Putzolu, I L Traiger, B W Wade, and V Watson. System R: Relational approach to database management. *ACM Transactions on Database Systems*, 1(2):97–137, 1976.

[Cab93] A M Cable. *Data Updates and Integrity Checking in Transformed Deductive Databases.* PhD thesis, The University of New Mexico, Albuquerque, NM, May 1993.

[CDV87] M J Carey, D J DeWitt, and S L Vandenburg. A data model and query language for EXODUS. Computer Sciences Technical Report 734, University of Wisconsin-Madison, Madison, WI, Dec 87

[CW87] D D Clark and D R Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, Apr 1987. IEEE Computer Society Press.

[GO90] D A Goldberg and A Orooji. A symmetrical approach to granting and revoking access rights in database management systems. In *Proceedings of the Fourteenth Annual International Computer Software and Applications Conference (COMPSAC)*, pages 124–131, Chicago, IL, Oct–Nov 1990. IEEE Computer Society Press.

[KS91] H F Korth and A Silberschatz. *Database System Concepts.* Advanced Computer Science Series. McGraw-Hill, New York, 1991.

[OvS92] M S Olivier and S H von Solms. Building a secure database using self-protecting objects. *Computers & Security*, 11(3):259–271, May 1992.

[Sjø92] D Sjøberg. Measuring schema evolution. Technical Report FIDE/92/36, University of Glasgow, 1992. ESPRIT BRA Project 3070.

SECURITY GUIDELINES FOR DATABASE SYSTEMS DEVELOPMENT

Prof. George Pangalos

Informatics laboratory, Computers Division, Faculty of Technology, General Department, Aristotelean University, Thessaloniki 540 06, Macedonia, Greece

Abstract

Database security plays an important role in the overall security of information systems and networks. A set of database security guidelines is described in this paper which aim to provide a functional guide for the introduction, administration and enforcement of the appropriate level of database security. The proposed guidelines, which have been developed in the framework of the SEISMED project of the EEC, help ensure the fulfilment of the corresponding set of security principles, as defined in the high level security policy. Each such principle is implemented through one or more of the proposed ten control agents, which specify the methodological means for satisfying one or more of these principles.

This work was supported in part by the Commission of the European Communities, AIM program, SEISMED (A2033) Project.

Keywords: database security guidelines, information systems security, database security.

1. INTRODUCTION

Security is an important issue when dealing with computer based information systems and networks (1,41,46,50). Current thinking in information systems security is that the issues center on confidentiality (information is only disclosed to those users who are authorized to have access to it), integrity (information is modified only by those users who have the right to do so), and availability (information and other IT resources can be accessed by authorized users when needed). Computer based information systems are 'risky' systems with respect to at least these issues. The level of security that should be included in a information system involves however some judgement about the dangers associated with the system and the resource implications of various means of avoiding or minimising those dangers. Developments in this field have progressed today to a point where information systems security needs to be tackled in a coherent and consistent fashion as a subject on its own right (7,41,44,50).

Database security in particular is an area of substantial interest in information systems security today. In addition to the more common security concerns of integrity, access control, audit etc., database systems add concerns for granularity, inference, aggregation, filtering, journaling etc. (1,41,50). Database systems also provide new tools for enforcing and controlling security. They also make it possible to increase granularity by enforcing security at a record or even at a data item level. The security of a single element may be thus different from the security of other elements of the same record or from values of the same attribute. That is, the security of one element may be different from that of other elements of the same database row or column. Database systems can also support several grades of security for sets of data or individual data items. These ranges may represent ranges of allowable knowledge, which may overlap (2,41).

This paper presents a set of database security guidelines for the development of a secure database system. The proposed guidelines help ensure the fulfilment of the corresponding set of security principles, as defined in the high level security policy of the specific establishment (47). Each such principle is implemented through one or more of the proposed ten control agents. The guidelines also provide, to the related categories of personnel, a functional guide for the introduction, administration and enforcement of the appropriate level of database security. Effort has been made to avoid technical details related to the methods and techniques used for the implementation of the guidelines, which are discussed in detail in (41).

2. DATABASE SYSTEMS SECURITY

Database security is concerned with the ability of the system to enforce a security policy governing the disclosure, modification or destruction of information. Within an organisation humans typically use a database as a technical tool for storing, processing and communicating information. At any time an amount of data has been stored in it, a large amount of messages has already been sent and the corresponding data can be called for duplication and further transmission on demand from potential receivers. The database relays the messages by persistently storing the corresponding data following the three phase procedure (2): "*accept messages ==> store / process data ==> assemble, duplicate and communicate data on demand*". The quality of mediation is dependably assured by special protocols enforcing completion of transactions and integrity constraints on stored data. Mediation is shared among many users and is required to be efficient in time and space (2).

Databases are usually an amalgamation of data from many sources; users entrust their data to a DBMS and rightfully expect protection of the data from unauthorised access, loss or damage. Databases contain structured data that are maintained by a database management system (DBMS) which is usually a separate software component that runs on the top of the operating system and provides the additional functions to use the database. It may also include functions to manage transactions. A DBMS assumes one or more data models upon which the data are structured (such as relations, networks,

hierarchies etc.) Database applications typically require a fine granularity of access control. From a security view point, database systems may be viewed as applications that require considerable kernel service or as protected subsystems and/or trusted processes. Databases may be considered to provide another level of security to complement that of the operating system.

The following assumptions about the database system environment and general principles related to database security have been widely accepted (10,11,14):

a. The database system security considerations must take into account all system S/W and H/W that touches information flowing into, and out of, the database. For example, an easily penetrated operating system would usually render a superbly protected DBMS useless.

b. Data integrity is a key requirement. The database system must preserve the integrity of the data stored in it. The user must be able to trust the system to give back the same data that is put in the system and to permit data to be modified only by authorised users. The data should not be destroyed or altered either accidentally, as in a system crash, or maliciously, as in some unauthorised person modifying the data. At the very least, the user should know if the data was corrupted.

c. Data should be available when needed. This implies system fault tolerance and redundancy in data, software and hardware.

d. Audit should be detailed enough to be useful and efficient enough so as not to severely burden system performance.

e. The aim should be to provide an adequate level of confidentiality (prevent disclosure) and yet preserve integrity by using appropriate concurrency and integrity controls (e.g. referential integrity).

f. The prototypes should be of general purpose, commercial quality and, according to most proposers, relational systems. The relational system has been chosen because it is (10) currently the model of preference in the commercial world.

Given the above definition and general framework of database security, we can regard a database as a channel in the sense of communication theory. Then a database security policy states (2) (i) which type of sub channels between (groups of) users can be established, (ii) the requirements of the availability of certain facilities of the sub channels, and (iii) the requirements on the (partial) separation and non-interference of sub channels. Seen from this point of view, we can identify two prominent proposals for database security policies (1,41,45):

a. The Mandatory security approach. The need for such a policy arises when a computer database system contains information with a variety of classifications and has some users which are not cleared for the highest classification of the information contained in the system. The approach is frequently based on the following assumption (constructs): there are users, data items and a lattice of security levels (1,41).

b. The discretionary security approach. Discretional access controls in today's database systems are designed to enforce a specific access control policy (1,41). The approach is based on the following assumption (constructs): there are users, (well informed) transactions, and (constraint) data items (1,2,41).

Each one offers a number of advantages. A basic distinction for example among the two is in the degree of protection they provide from Trojan horse programs. In general discretional security is set by the user and can be defeated by Trojan horse programs, while mandatory security is set by the database system and is much more effective against Trojan horse programs.

3. THE SEISMED PROJECT

The work reported in this paper has been based on research undertaken in the framework of the SEISMED project of the European Union (EU). The SEISMED project (a Secure Environment for Information Systems in MEDicine) is part of the European Union's AIM (Advance Informatics in Medicine) program. AIM is currently investing some 90 milion

ECU to provide opportunities for improving computer information systems across Europe within the medical environment. The project lasts for three years (1992-94) and is implemented by a European consortium (41). It is the only one in the area of information systems security and one of the biggest of the program.

The main objectives of SEISMED are:
- To develop a High Level Security Policy to enable organizations using information systems to follow a consistent path;
- To develop specific guidelines to enchance security in existing systems, in the design and implementation of future systems, and in systems using networks;
- To develop an encryption prototype for use in health care environments;
- To perform risk analyses at a number of health care centers across Europe to identify the opportunities and needs for improved security;
- To examine, across Europe, the legal issues of data protection and privacy with health care information systems in order to develop a common deontology or code of ethics.
- To identify mechanisms by which the results of SEISMED can be put in effective use.

The technical approach used in SEISMED was to break the project into a number of inter-connecting themes, which are:
- The identification of current practices by means of a survey throughout Europe and detailed risk analyses at four healthcare centres;
- The preparation of guidelines detailing:
  - the development and implementayion of a high level security policy
  - how to perform risk analysis
  - how to include security with a system's design
  - how to retrospectively include security within existing systems
  - how to achieve security where networks are utilised
  - the use of encryption in health care environments
  - the legal framework across the European Union countries.
- To test the implementation of these guidelines (except for the legal framework) by the reference centres participating in the project.
- To revise the guidelines in light of the reference centre experiences.

4. SECURE DATABASE DEVELOPMENT METHODOLOGY

The problem of developing a secure database system consists of three main issues (3,4,16):
(i) the **definition** of the semantics of the secure database to be developed, that is to characterise the needed security properties in terms of the database semantics,
(ii) the **implementation** of those semantics on a database system, that is on a DBMS and on the data it handles, and
(iii) **assuring** that the implemented system provides the needed security properties.

A development methodology has the purpose of specifying how each one of these three secure database development issues can be achieved. This is usually accomplished by guiding the various steps of the development, by providing modelling and analysis tools, and by organising these three issues into a global framework allowing to achieve consistency of the whole development process and of the target system. Such a development methodology should be multiphase (29) in order to allow for incremental construction of the secure database system. The various phases of the methodology should also be supported by models able to represent the details necessary to each phase and to support the types of analysis about the system under design which pertain to each phase. Multiphase development of systems is a widely-accepted principle today. The benefits of a multiphase approach lie mainly in the separation between design and implementation aspects; this is the principle of security policies-mechanisms separation (29).

Although several information system development methodologies have been described in the literature, the structure of most of them is based on the same traditional generic ("V", or "waterfall") model, which includes the following phases (49): Requirements Analysis (REQ), Prototyping (PRO), Design Specifications (DES), Coding (COD), Testing (TES), and Verification (VER). In line with the above, database development could also be seen that is generally carried out in a number of similar steps (41): Database system security requirements analysis (REQ), Prototyping (PRO), Database system design specifications (which include conceptual, logical and physical design, DES), Coding (COD), Database system testing (TES), and Database system verification (VER). Such a database design methodology has been described in (41).

It is important to note that since the database system is part of the overall information system, database security should be seen as an integral part of the overall information system security. Security aspects must therefore be considered in a unified way as early as possible in the information systems design and implementation process. Prior to the definition and implementation of database security, a well defined overall security policy and a suitable overall system design methodology are therefore required. Such a detailed methodology and a set of guidelines for the development of a suitable high level security policy has also been developed in the framework of the SEISMED project of the EEC and can be found in (47).

If the above development methodology is to be used for the development of a security-sensitive database system, then specific security guidelines, like the ones proposed in later chapters, should be included in the description of all these phases (5,6,41,47). The guidelines help ensure that each of the database development phases will be performed according to existing security standards and regulations. They help also to provide, to the related categories of personnel, a functional guide for the introduction, administration and enforcement of the appropriate level of database security. The database security guidelines have the following characteristics:
(i) Each guideline intends to fulfil at least one of the requirements for database security,
(ii) Each guideline addresses at least one personnel category, and
(iii) They are based on the requirements for database security and the personnel duties and responsibilities, the potential and limitations of information systems and database technology today, and the establishment's organisational structure and procedures.

5. THE CONTROL AGENTS

The guidelines proposed herein are provided for the fulfilment of a set of predefined (in the high level security policy (HLSP)) database security principles (47,49). Each principle is implemented through one or more control agents. The **control agents** provide the means for satisfying one or more of the basic security principles (49). The proposed methodology incorporates ten such control agents, namely: tools, review, documentation, formalism, traceability, standardization, code reuse, methodology, responsibility and reliability. Their scope is described in the sequel.

1. Tools (TLS):

Software products should be employed by the database system designer to ensure that every step in the development process will be carried out according to efficiency or security standards. Example: Tools that measure complexity and potential programming language violations

2. Review (RVW):

All system development phases should be reviewed by teams of security experts together with users and relevant technical experts, to ensure that all procedures have been followed according to security and integrity standards and criteria. Example: Test reviewing should be used to ensure that all test procedures have been carried out properly

3. Documentation (DOC):

Documentation should follow the end of every development phase and it should be used as a reference to the security actions that should have been performed that far. Example:

Documentation of the source code of all the database application programs can help for comparing the programs that should run, with the ones that actually do. This control agent forms the main input to the verification process at each stage as well as to the validation and certification processes.

4. Formalism (FRM):

Specific formal procedures should be followed in phases where the complexity of the development may cause misinterpretation or flaw of information, or other relevant inconsistencies. Example: The conceptual database design should be described in a formal specification framework.

5. Traceability (TRC):

In several cases capability of tracing back a procedure to the requirement which has originated this procedure should exist. Example: Every application program should be associated with its functional requirement(s).

6. Standardisation (STD):

Standards help the database system developer to employ a uniform set of approaches, thus decreasing the types of techniques that are available for malicious software developers. Example: Coding standards in database applications programming increase the auditability and maintainability of the resulting code.

7. Code Reuse (CRU):

Code reuse provides a means of error propagation; therefore it should be done according to specific security procedures. Example: If prototype code is reused in the development code, then it should be first tested and verified towards well defined security goals.

8. Methodology (MET):

Specific methodologies should be used to ensure that a procedure has been performed in the (secure) way that it should be. Example: All test tasks should be performed according to a methodology, capable of ensuring that all security critical test data have been used.

9. Responsibility (RES):

The nomination of person(s) or organisation(s) who is(are) responsible for the fulfilment of all security goals is an effective means toward preventing security violations. Example: The responsibility for the verification tasks should not be placed with the body responsible for the database testing tasks.

10. Reliability (REL):

The reliability of some specific procedures should be measured and used to reduce unwanted side-effects to an acceptable level. Example: Test results should be used to reduce observed software flaw densities.

6. GUIDELINES DESCRIPTION

Now we present a structured description of a set of database security guidelines which help ensure the fulfilment of the corresponding set of information system security principles (as defined in the HLSP for the specific establishment, (41)). Each such principle is implemented through one or more of the ten control agents described earlier. As discussed in (49), information systems developed according to the provisions set out in the proposed guidelines enjoy security-related functions efficient and integratable with all other specified functions. Given the scope of this paper, this list cannot be exhaustive. Part of the security guidelines that we have proposed in the framework of the SEISMED project of the EEC (41) is set out below in order to demonstrate the process (41,47). We have tried to avoid in this description technical details related to the implementation of the guidelines. A detailed discussion and presentation of methods and procedures for the implementation of the proposed guidelines can be found in (41).

We can classify database security guidelines into three main categories: (i) Database development guidelines, (ii) Control of database software guidelines, and (iii) Database operational and organisational guidelines. The actual description of each guideline below is preceded by a header conforming to the following format:

Guideline short title [Code Number /Phase / Control Agent]

In this format:

- **"Phase"**:

The database development phases include the ones described earlier, and are refered to as follows:

i. Database development security guidelines:
- Preliminary analysis security guidelines ---> (PRE)
- Database system security requirements analysis ---> (REQ)
- Prototyping (where applicable) ---> (PRO)
- Database system design specifications ---> (DES)
(which include conceptual, logical and physical design)
- Coding (e.g. using DBMS tools) ---> (COD)
- Database system testing ---> (TES)
- Database system verification ---> (VER).

ii. Control of database software security guidelines:
- General precautions ---> (GPR)
- Software development ---> (SDV)

iii. Operational and organisational security guidelines:
- Database organisational/administration guidelines ---> (ORG)
- Database operational guidelines ---> (OPER)

("A" in the development phase stands for "All phases").

- **"Control agent"**:

It refers to the control agent codes described in the previous section as follows:

1. Tools ---> TLS
2. Review ---> RVW
3. Documentation ---> DOC
4. Formalism ---> FRM
5. Traceability ---> TRC
6. Standardisation ---> STD
7. Code Reuse ---> CRU
8. Methodology ---> MET
9. Responsibility ---> RES
10. Reliability ---> REL

("OP" in the control target stands for "Overall Philosophy"),

7. DATABASE SECURITY GUIDELINES

7.10 DATABASE DEVELOPMENT - I

The set of the proposed security guidelines to be followed in **all phases** of the database development process is the following:

1. *Integrated Design Methodology [DBG610-1/A/OP]*
Data protection procedures should be specified along with the information system's development specifications.
2. *Auditing [DBG610-2/A/OP]*
A record of all controlled software development operations should be logged and stored by the software development department in a protected repository.
3. *Mediation [DBG610-3/A/OP]*
All controlled development operations should be automatically mediated by the software development department with respect to an explicit mandatory and discretionary security regulations.
4. *Trusted Path [DBG610-4/A/OP]*
An explicit mechanism should be included in the development department to ensure that all controlled software development operations cannot be imitated or intercepted by unauthorised means.
5. *Identification - Authentication [DBG610-5/A/OP]*

No developer should initiate or participate in any development operation unless they have first been identified and authenticated by the software development department responsible person.

6. *Configuration Management [DBG610-6/A/OP]*
   All software resulting from controlled software development activity should be stored in a protected repository that maintains and controls all software versions, software modification requests, software changes, and related information such as modification request initiators and change responsibility.
7. *Environment Integrity [DBG610-7/A/OP]*
   An explicit procedure should be available for identifying changes in all the environment and tool-related software and, if required, to restore the integrity associated with that software.
8. *Trusted Distribution [DBG610-8/A/OP]*
   All software should be delivered to the departments of the establishment in a manner that ensures that the integrity has not been compromised.
9. *Intrusion Detection [DBG610-9/A/OP]*
   Audit records should be used to perform periodic and random intrusion detection analysis on the software development department.
10. *Administration [DBG610-10/A/OP]*
    The software development department, as well as the tool software and the developed application code should be maintained according to explicit administrative documentation by experienced administrators.
11. *Environment and Tools [DBG610-11/A/TLS]*
    All tool software should be selected according to an explicit selection policy that considers the maturity and development course of the software.
12. *Least Privilege [DBG610-12/A/OP]*
    Privileges to perform controlled software development operations should be allocated and maintained so that a privilege is only given to establishment personnel individuals who require that privilege.
13. *Multi-person Control [DBG610-13/A/OP]*
    Controlled development operations should be completed with the active endorsement and involvement of more than one experienced software developers.
14. *Security Policy [DBG610-14/A/OP]*
    All controlled software development operations should be performed in accordance with an explicitly defined and enforced security policy.
15. *Shared Knowledge [DBG610-15/A/OP]*
    At least two experienced individuals should be thoroughly familiar with all aspects of the requirements and design.
16. *Software Reuse [DBG610-16/A/CRU]*
    Any software reused during development should be selected according to an explicit reuse policy that takes into account maturity and facilitates for building object code from reused source.
17. *Planning [DBG610-17/A/OP]*
    The characteristics of all development activities should be described in a Development Plan (DP) and the management of the software development should follow the approach described in this SDP.
18. *Risk Mitigation [DBG610-18/A/OP]*

All potential risks associated with the development approach should be explicitly identified and a risk mitigation strategy should be documented and complied with throughout the software development life cycle.

7.20 DATABASE DEVELOPMENT - 2

The set of the proposed security guidelines **for each specific phase** of the design and implementation of databases is the following:

i. Preliminary analysis phase:

1. *Risk analysis [DBG620-1-1/PRE/MET]*

Risk analysis of the target system should be performed prior to any database development steps. Risc analysis should include assets identification, identification of vulnerabilities of assets, estimation of the likelihood of exploitation, estimation of expected costs, survey of new controls, savings estimation etc.

ii. Database system requirements analysis phase:

1. Requirements Analysis Tools [DBG620-2-1/REQ/TLS]
Automated requirements analysis tools should be employed in order to provide requirements specification, consistency checking, and documentation generation.

2. Requirements Analysis Review [DBG620-2-2/REQ/RVW]
Requirements analysis activities should be performed by multiple experienced development personnel, at least one of whom is not directly involved in requirements analysis.

3. Requirements Analysis Documentation [DBG620-2-3/REQ/DOC]
The characteristics of the requirements analysis process employed and a rationale for all requirements, should be documented

4 Formal Requirements Specifications [DBG620-2-4/REQ/FRM]
Requirements should be specified in a formal specification framework.

5. Requirements Traceability [DBG620-2-4/REQ/TRC]
All requirements should be directly traceable to an explicit user source.

iii. Prototyping phase:

1. Prototyping Approach [DBG620-3-1/PRO/MET]
Prototyping should be performed according to an explicitly defined prototype plan that describes the manner in which the prototype is designed, developed, tested, documented, and protected.

2. Prototype Code Reuse [DBG620-3-2/PRO/CRU]
If prototype code is reused in the developed code, then the prototyping approach should be taken into account in the measurement of software trust.

iv. Database system design specification phase (includes conceptual, logical and physical design):

1. Design Tools [DBG620-4-1/DES/TLS]
Design tools should be employed to maintain design requirements traceability mappings and to generate design documentation.

2. Design Review [DBG620-4-2/DES/RVW]
All design decisions should be reviewed by multiple experienced software development personnel, at least one of whom is not directly involved in the design.

3. Design Documentation [DBG620-4-3/DES/DOC]
The characteristics of the design process, design alternatives considered, and design rationales should be documented

4. Formal Specification [DBG620-4-4/DES/FRM]
The design should be specified in a formal specification framework.

5 Design Traceability [DBG620-4-5/DES/TRC]
All aspects of the design should be demonstrated in all design documentation to be traceable to an explicit set of requirements.

v. Coding phase:

1. Coding Standards [DBG620-5-1/COD/STD]
An explicitly defined coding standard that enforces modular, structured programming should be complied with throughout the coding activity.

2. Code Analysis Tools [DBG620-5-2/COD/TLS]
All developed code should be subjected to code analysis using tools that measure complexity, style, and potential programming violations.

3. Code Review [DBG620-5-3/COD/RVW]
All code should be reviewed by multiple experienced software development personnel, at least one of whom is not directly involved in coding.

4. *Code Documentation [DBG620-5-4/COD/DOC]*
The characteristics of the coding process, the module organisation, criteria used for module decomposition, and source code comments, should be documented.
5. *Code Traceability [DBG620-5-5/COD/TRC]*
All code should be shown in the source code documentation to be directly traceable to an aspect of the design and to a set of requirements.

vi. Database system testing phase:

1. *Test Strategies [DBG620-6-1/TES/MET]*
All test and integration tasks should include provision for security, functional, penetration, regression, and random testing.
2. *Test Responsibility [DBG620-6-2/TES/RES]*
The responsibility for testing should be placed with an independent body, not directly involved with coding or design.
3. *Reliability Measurement [DBG620-6-3/TES/REL]*
Test results should be used to reduce observed software flaw densities to an acceptable level.
4. *Testing Tools [DBG620-6-4/TES/TLS]*
The software development department should include an automated testbed for creating, executing, documenting, and analysing the completeness of all tests.
5. *Test Review [DBG620-6-5/TES/RVW]*
All tests should be reviewed by multiple experienced software development personnel, at least one of whom is not directly involved in testing.
6. *Test Documentation [DBG620-6-6/TES/DOC]*
The characteristics of the test process should be documented.
7. *Test Traceability [DBG620-6-7/TES/TRC]*
All tests should be shown in the test documentation to be directly traceable to explicit aspects of the code, design, and requirements.

vii. Database system verification phase:

1. *Design Verification [DBG620-7-1/VER/FRM]*
A formal verification should be performed to prove that the formal design specification correctly meets its requirements.
2. *Code Verification [DBG620-7-2/VER/FRM]*
A formal verification should be performed to prove that a low-level formal specification of the code correctly meets its requirements.
3. *Verification Responsibility [DBG620-7-3/VER/RES]*
The responsibility for verification should not be placed with the organisation responsible for testing.
4. *Verification Tools [DBG620-7-4/VER/TLS]*
Design and code verifications should be performed with the assistance of an automated verification environment.
5. *Verification Review [DBG620-7-5/VER/RVW]*
All verification results should be reviewed by multiple experienced software development personnel, at least one of whom is not directly involved in verification.
6. *Verification Documentation [DBG620-7-6/VER/RVW]*
The design and code verification processes employed and all assumptions required to interpret the verification results should be documented explicitly.
7. *Verification Condition Traceability [DBG620-7-7/VER/TRC]*
All verification conditions should be shown in the verification documentation to be traceable to explicit aspects of the requirements, design and code.

7.30 CONTROL OF DATABASE SOFTWARE

i. General precautions

1. Demarcation *[DBG630-1-1/GPR/MET]*
A strict demarcation must be maintained between the operational versions of database software and programs and their test equivalent. These should be kept in

separate libraries. The operational libraries must exist in both source and object form. The operational form library should be used for the live operations and be called up as necessary by the users for on-line or batch operations. The operational source library should exist only as a means of support for the object library. Database programmers and other development staff should never normally access the operational library.

2 New systems *[DBG630-1-2/GPR/MET]*

New systems and changes to live programs will be lodged on the test source library *(DBG630-1-1)*. Only when the tests on the new version are completed, will it be transferred from the test to the operational libraries. This must always be done through a formal procedure.

3 Operational versions [DBG630-1-3/GPR/MET]

A copy of the operational version of the source code of each operational database program should be kept until any new version has been established as working correctly.

4. Back up [DBG630-1-4/GPR/MET]

Back up of database software is as important as the back up of the database data. Copies of database software should include both operational and test libraries.

5 Change log [DBG630-1-5/GPR/MET]

A log should be kept of all changes to the operational software libraries. Usually this will consist of all the formal authorities to transfer, kept in a file. Sufficient details of why any changes were made and the date of transfer should be recorded.

II Software development

1. DB software specifications [DBG630-2-1/SDV/FRM]

All database software written in-house should be written from a full specification. When the specification is first drawn up it should be agreed by the database users concerned, or their representative, and signed as correct. If there is any financial impact, the database auditor should also agree the specification and add any controls that are necessary from the audit point of view. Checks should be made during development on compliance with the full specification and the establishment's security policy.

2 DB software testing [DBG630-2-2/SDV/MET]

All database software written in-house should be thoroughly tested. The testing may consist of several phases. For example: the programmer testing that the software does what is expected, the database analyst testing that the performance matches his own expectations, the supervision by another programming specialist to ensure that all paths in the software have been tested, the checking of the results by the end users, the checking of the results by the database auditor (if anything financial is involved), etc.

3 DB software maintenance [DBG630-2-3/SDV/MET]

As much care should be taken over maintenance of database software as for its development. The same standards of testing, checking, documentation and sign-off should apply to maintenance as to pure development.

From the security point of view, we may be think of three classes of maintenance, with the corresponding security guidelines: corrective, adaptive and enchancing. Corrective maintenance is required when the software is found to be not in keeping with the specification or the real needs of the database users, or when an error has been discovered. Adaptive maintenance is required when database software needs to be adapted for changed circumstances while basically performing the same function. Finally, enhancing maintenance is required when new uses or better services are desired and the database software is adopted to provide them.

4. Corrective maintenance [DBG630-2-4/SDV/STD]

It is important that corrective maintenance takes place and errors are corrected as soon as possible. There is no need for end users or the database auditor to be involved, as the maintenance is only required to provide what should have been there in the first place. However, throughout testing is required and the results

should be tested by someone else other than the programmer changing the software. A log must be kept of the change and the data and time that it was effective in case of errors discovered later.

5. Adaptive and enchancing maintenance [DBG630-2-5/SDV/STD]
   When maintenance is required for those reasons, more formal procedures should be followed. For example, the originator of the maintenance (e.g. the database user) should be required to complete an amendment request, which must be agreed by the database auditor.
6. Copies [DBG630-2-6/SDV/MET]
   Measures must be taken so that where multiple copies of database software are in use, they remain consistent. End users should only be permitted to maintain their own applications software after specific agreement with the database administrator, and only if they hold the single instance of this software and they are able to take responsibility for any changes.

7.40 DATABASE ORGANISATION AND OPERATION

i. Database organisational and administration guidelines

1. Scope [DBG640-1-1/ORG/OP]
   Database system security considerations should take into account all system software and hardware that touches information flowing into, and out of, the database. Example: an easily penetrated operating system would render a superbly protected database management system useless.
2. Database security policy [DBG640-1-2/ORG/MET]
   A well defined database security policy should be established before the database goes to operation. The database security policy should provide adequate guidance on issues like: database security policy administration, database security policy specification, database access control specification, database information flow control, enforcement of control, etc..
3. Database administration [DBG640-1-3/ORG/OP]
   The position of a database administrator should be filled prior to the implementation of the database. Appropriate decisions on his selection criteria, duties and responsibilities should be taken in the early stages of database development.
4. Access control [DBG640-1-4/ORG/OP]
   A detailed access control policy should be specified, so that database users are allowed to access only authorised data and so that different users can be restricted to different modes of access. The database access policy should include: database access control administration, database access control specification, database access control enforcement, etc..
5. Integrity [DBG640-1-5/ORG/OP]
   Data integrity is a key requirement. The integrity of the database should be maintained so that the database data are immune to physical problems, the structure of the database is preserved and the data contained in each element is accurate. The user must be able to trust the system to give back the same data that is put in the system and to permit data to be modified only by authorised users. At the very least, the user should know if the data was corrupted.
6. Confidentiality [DBG640-1-6/ORG/OP]
   The database system should provide an adequate level of confidentiality/secrecy (prevent unauthorised disclosure) and yet preserve availability and integrity by using appropriate controls (e.g. referential integrity).
7. Availability [DBG640-1-7/ORG/OP]
   Database users should be able to access all the database data for which they are authorised. This implies system fault tolerance and redundancy in data, software and hardware. Inference and aggregation must be studied and controlled. A specific availability policy should be developed which will deal with problems like arbitrating two users' requests for the same data item, withholding some non-protected data in order to avoid revealing protected data, etc..
8. People [DBG640-1-8/ORG/OP]

People are crucial in security. Trusted individuals, such as operators and programmers, must be carefully selected because of their potential ability to affect the computer (database) system and all computer users. Well defined procedures and rules for employee selection must be developed and used.

9 Education and awareness [DBG640-1-9/ORG/OP]

Suitable training programs and seminars must be planned for all types of database users. Education and awareness programs must take place on a periodical basis. There should be different such programs for every major type of users (e.g. , technical, administration users, etc.).

ii Database operational guidelines

1 User authentication [DBG640-2-1/OPER/OP]

The DBMS should be designed to perform its own authentication at the required level, in addition to the authentication performed by the operating system. Specific authentication procedures should be implemented according to the specific needs and security requirements of the particular establishment.

2 Auditability [DBG640-2-2/OPER/OP]

Appropriate audit procedures should be developed so that it will be possible to track unauthorised accesses or modifications of the elements of the database. Audit should be detailed enough to be useful and sufficient enough so as not to severely burden the system performance.

3 Inference control [DBG640-2-3/OPER/OP]

An inference prevention policy should be specified. This should include issues like: database inference prevention administration, database inference prevention specification, database inference prevention control, etc.

4 Database recovery [DBG640-2-4/OPER/OP]

A detailed database recovery policy should be specified. This policy should define: database recovery prevention procedures, database recovery administration, database recovery specification, database recovery control, etc..

8. REFERENCES

1 Lunt T., Security in database systems, Computers and security journal, Vol 7,No.1, 1992

2. Biskup J., Medical database security, in data protection and confidentiality in health informatics, EEC DGXII ed.,IOS press, 1991.

3 Landwehr C., ed., Database security II: Status and prospects, North-Holland, 1989.

4 Spooner D., Landwehr C., eds., Database security III, North-Holland, 1990.

5. Proceedings ESORICS (European Symposium on Research in Computer Security), Toulouse, France, 1990.

6 Jajodia S., Landwehr C., eds., Database security IV, North-Holland, 1991.

7 EEC DGXII, ed., Data protection and confidentiality in health informatics, IOS press, 1991

8 Biskup J., Analysis of the privacy model for the information system DORIS, in (3).

9 Camnatac A., Data protection issues in database management and expert systems, in (7)

10 Campbell J., A research and development program for trusted distribute DBMSs, in Database security IV, Jaodia (ed), North Holland, 1991

11 DoD, Department of Defence Trusted computer system evaluation criteria, DoD 5200.28-STD, 1985

12. National Computer Security Centre, Draft trusted DBMS interpretation of the DoD trusted computer system evaluation criteria, USA, 1989

13. National Computer Security Centre, Trusted network interpretation of the trusted computer system evaluation criteria, NCSC-TG-005, USA, 1987.

14. Information Technology Evaluation Criteria (ITSEC), Version 1.2, EEC Document, Brussels, June 1991.

15. Information Technology Security Evaluation Manual (ITSEM), Draft V0.2, EEC Draft Document, April 1992.

16. Landwehr C. E., Minutes of IFIP-TC11 1986 meeting, Montecarlo, December 1986.
17. Stonabraker M., The design and implementation of INGRES, ACM TODS, Vol. 1, No. 3, 1976.
18. Zloof M., Query by example: a database language, IBM systems Journal, Vol. 16, No. 4, 1977.
19. Astrahan M., System R: Relational approach to database management, ACM TODS, Vol. 1, No. 2, June 1976.
20. McGee W., The information Management System IMS/VS. Part V: Transaction processing facilities, IBM systems journal, Vol. 16, No. 2, 1977.
21. Landwehr C., The best available technologies for computer security, IEEE Computer, Vol. 16, No. 7, 1983.
22. ACF2: The access control facility - General information manual, 1983.
23. Secure product description. Bull and Babbage publ., 1979.
24. Duffy K. and Sullivan J., Integrity lock prototype, in the Proceedings 4th IFIP international security conference, Montecarlo, 1986.
25. Cerniglia C. and Millen J., Computer security models, MTR project, Report No. 9531, 1984.
26. Landwehr C., Formal models for computer security, ACM computer surveys, Vol. 13, No. 3, 1981.
27. Griffiths P. and Wade B., An authorisation mechanism for a relational database system, ACM TODS, Vol. 1, No. 3, 1976.
28. Fagin R., On an authorisation mechanism, ACN TODS, Vol. 3, No. 3, 1976.
29. Fugini M., Secure database development methodologies, in (3)
30. Dwyer P., Multilevel security in database management systems, Computers and security, Vol. 6, No. 3, 1987.
31. Akl S., Views for multilevel database database security, IEEE Trans. on S/W Eng., Vol. 13, No. 2, 1987.
32. Hartson H., Database security - system architectures, Information systems, Vol. 6, N0.1, 1981.
33. Leveson J., Safety analysis using Petri nets, IEEE Trans. on S/W Eng., Vol. 13, No. 3, 1987.
34. Bussolati U., A database approach to modelling and managing of security information, Proc. 7th Int. Conf. on VLDB, Cannes, 1981.
35. Bussolati U., Data security management in distributed databases, Information systems, Vol. 7, No. 3, 1982.
36. Date C., An introduction to database systems, Vol. 2, second ed., Addison-Wesley, 1986.
37. Ting T., Application information security semantics: A case of mental health delivery, in (4).
38. Hinke T., DBMS trusted computing base taxonomy, in (4).
39. Graubart R., A comparison of three secure DBMS architectures, in (4).
40. Hosmer H., Designing multilevel secure distributed databases, in (3).
41. Pangalos G., Security in medical database systems, EEC, SEISMED project report, No. INT S.3/92, 1992.
42. J.V. Marel, A.B. Bakker, User accessrights in an intergrated hospital information system, IFIP-IMIA, North-Holland, 1988.
43. J. BisKup, A general framework for database security, Proc. EROSICS, Toulouse, France, 1990, pp. 35-41.
44. J. Biskup, Medical database security, Proc. GI-20, Jahrestagung II, Stutgart, October 1990, Springer-Verlag, 1990, pp. 212-221.
45. T.C. Ting, S.A. Demurjian, M.Y. Hu, A specification methodology for user-role based security in an object-oriented design model, Proc. 6th IFIP WG11.3 on database security, 1993.
46. C. Pfleeger, Security in computing, Prentice hall, 1991.
47. S. Katsikas, D. Gritzalis, High level security policies, SEISMED report, June 1993.
48. S. Oliver, S., Building a secure database using self-protecting objects, computer security journal, vol. 11, no. 3, pp. 259-71.

49. Gritzalis, D., Katsikas, S., Pangalos, G., Design of medical information systems, Proceedings: MIE'93 conference, Israel, 1993.
50. Lunt T., Research directions in database systems, Springer-Verlag, 1992.

ACKNOWLEDGEMENT
I would like to thank Mr. John McDermont for his interest and constructive comments.

Verzeichnis der Hildesheimer Informatik-Berichte

1/94 K. Diethelm:
Modified Interpolatory Quadrature Rules for Cauchy Principal Value Integrals
(Januar 1994)

2/94 S. Ehrich:
On the Error of Extended Gaussian Quadrature Formulae for Functions of Bounded Variation
(Januar 1994)

3/94 S. Ehrich:
Asymptotic Behaviour of Stieltjes Polynomials for Ultraspherical Weight Functions
(Februar 1994)

4/94 K. J. Förster:
On the Weights of Positive Quadrature Formulas for Ultraspherical Weight Functions
(April 1994)

5/94 K. J. Förster:
Inequalities for the Weights of Positive Quadrature Formulas for the Legendre Weight Function
(April 1994)

6/94 Institut für Mathematik:
Vorträge auf der Jahrestagung der GAMM vom 4.4. - 8.4.1994 in Braunschweig
(April 1994)

7/94 K.-J. Förster, P. Köhler, G. Nikolov:
Monotonicity and Stopping Rules for Compound Gauss-Type Quadrature Formulae
(April 1994)

8/94 U. Gehrmann:
Eine Methode für Entwurf und Spezifikation von Protokollkonvertern
(April 1994)

9/94 K. Diethelm:
A Definiteness Criterion for Linear Functionals and its Application to Cauchy Principal Value Quadrature
(April 1994)

10/94 J. Biskup, G. Bleumer:
Reflections on the Security of Database and Datatransfer Systems in Health Care;
J. Biskup:
Impacts of Creating, Implementing and Using Formal Languages
(April 1994)

11/94 B. Pfitzmann, M. Waidner:
A General Framework for Formal Notions of "Secure" Systems
(April 1994)

12/94 M. Hagström:
Error Tolerant Retrieval in Large Text Files
(Mai 1994)

13/94 M. Heiland
Using Data Compression for the Representation of Sparsely Coded Associative Memory
(Mai 1994)

14/94 H. J. Bentz, Jürgen Braun:
Über die Werte von Zeta(2n+1)
(Mai 1994)

15/94 H. J. Bentz, Jürgen Braun:
Some Strange Series Involving π
(Mai 1994)

16/94 B. Pfitzmann
Fail-Stop Signatures Without Trees
(Juni 1994)

17/94 J. Biskup, R. Menzel, T. Polle
Transforming an Entity-Relationship Schema into Object-Oriented Database Schemas
(Juni 1994)

18/94 D. Kadelka, M. Kovonko, S. Voss
Epistasis Variance in Genetic Algorithms
(Juli 1994)

19/94 K. Diethelm,
Asymptotically Sharp Error Bounds for a Quadrature Rule for Cauchy Principal Value Integrals Based on Piecewise Linear Interpolation
(Juli 1994)

20/94 J. Biskup, M. Morgenstern, C. Landwehr (Editors)
Proceedings of the IFIP WG 11.3 Eighth Annual Working Conference on Database Security
(Juli 1994)