AD-A236 657

(2)

USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

ARMY NONPROGRAMMER SYSTEM FOR WORKING ENCYCLOPEDIA REQUESTS PHASE III-B REPORT (ASQB-GI-91-025)

APRIL 1991

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188 Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ASQB-GI-91-025	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION AIRMICS — 6b. OFFICE SYMBOL *(If applicable)* ASQB-GI	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS (City, State, and Zip Code)	7b. ADDRESS *(City, State, and ZIP Code)* N/A
8b. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS — 8b. OFFICE SYMBOL *(If applicable)* ASQB-GI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Army Nonprogrammer System for Working Encyclopedia Requests, Phase III-B Report

12. PERSONAL AUTHOR(S)
Dr. Karen Ryan, Honeywell Inc.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical Paper	FROM Oct 90 TO Mar 91	April 12, 1991	46

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Heterogeneous Databases, Distributed Processing, SAILS, SAACONS, SQL Query, Data Management, Scheme Integration, Database Security

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report details the research efforts into the access of distributed heterogeneous databases through an encyclopedia facility. Specifically several data management tools have been prototyped and are described. Current capabilities include database schema registration, schema integration, graphical browser, query formulation and query processing. Two subschemas of the SAILS and SAACONS databases provided by the PM-ISM were implemented and used for system testing. A study on the security issues of integrating and accessing multiple standalone databases as a single integrated database was conducted and these results are also reported in this report.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL MAJ(P) David S. Stevens	22b. TELEPHONE *(Include Area Code)* (404) 894-3110 — 22c. OFFICE SYMBOL ASQB-GI

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

This research was performed by Honeywell Federal Systems, contract number DAKF11-88-C-0024, for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). The report discusses a set of data management tools and distributed query processing modules developed and/or enhanced during a 6 month effort, the third phase of a four phase effort. Requests to view a demonstration of the prototype may be made by contacting MAJ(P) David S. Stevens at 404/894-3110. This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

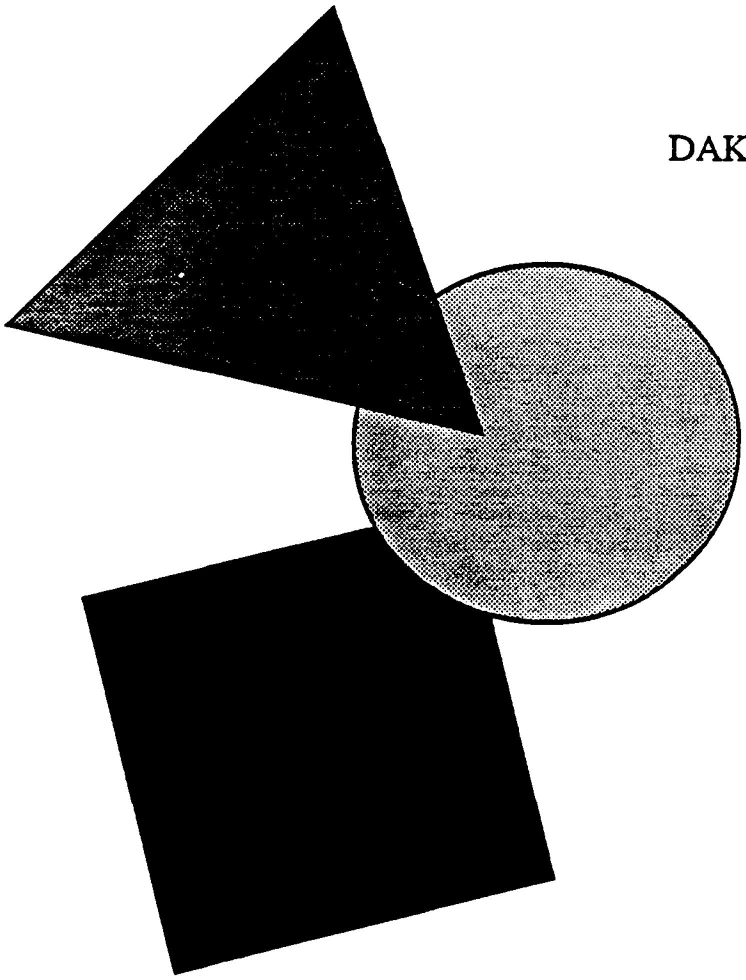
THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ [illegible]
Glenn E. Racine, Chief
Computer and Information
Systems Division

s/ [illegible]
John R. Mitchell
Director
AIRMICS

HFS Inc.

ANSWER Phase IIIB Final Report



Contract No.
DAKF11-88-C-0024

April 12, 1991

ANSWER
Phase IIIB Final Report

Contract No. DAKF11-88-C-0024

Prepared for:

AIRMICS

HFS Inc.
7900 Westpark Drive
McLean, VA 22102

April 12, 1991

Table of Contents

Table of Contents

List of Figures

List of Tables

Section 1
Introduction

This report covers the Phase IIIB activities of the ANSWER program from October 1, 1990, through March 30, 1991.

1.1 Phases I, II and IIIA

In Phases I, II and IIIA of the ANSWER program, we developed several prototype data administration tools to support the creation and maintenance of heterogeneous distributed database systems. We also designed and implemented query decomposition algorithms that accept a query stated against an integrated schema, decompose the query into queries against individual schemas, execute the queries and recompose the results. We also designed a flexible SQL query formulation tool that supports creation of queries using either schema terms or standard data element terms. The overall ANSWER architecture as envisioned as of the end of the program is shown in Figure 1-1. Deliverables for Phases I, II and IIIA are highlighted.

The schema integrator creates integrated schemas from individual schemas and data administrator assertions about individual schema relationships. The integrated schemas can be used as the basis of distributed query processing by providing a global definition of available data and static mappings to local systems. The schema integrator reads and writes schemas to and from an IRDS repository, an ANSI and FIPS standard for data definition repositories. All ANSWER tools will eventually share information through the IRDS repository.

The browser and schema editing facilities provide a graphical browser for entity-category-relationship schemas and for information about standard data elements. The browser/editor also reads information from and writes information to IRDS. The editor supports the graphical creation of schemas and the inspection of associated standard data element information. The browser also supports the browsing of integrated schema information and associated individual schema information.

The Data Element Creation Tool (DECT) is designed to support the creation of standard data elements. The tool is intended to be used at the installation level as an aid to developing new standard data element names. It supports the creation of a syntactically correct name (in accordance with AR 25-9) as well as the semantic analysis of that name to determine if any other closely related names have already been proposed. The analysis is based on an internal representation of the semantic structure of the data element name and may be related to the use of that name in a schema.

The User Interface Manager is a multiple-window graphical user interface that supports control integration of multiple tools. Each tool is represented as a state transition diagram that is managed by the interface manager. The state transition diagram controls the points at which the user may suspend execution of one tool and invoke execution of a different tool. The user interface manager

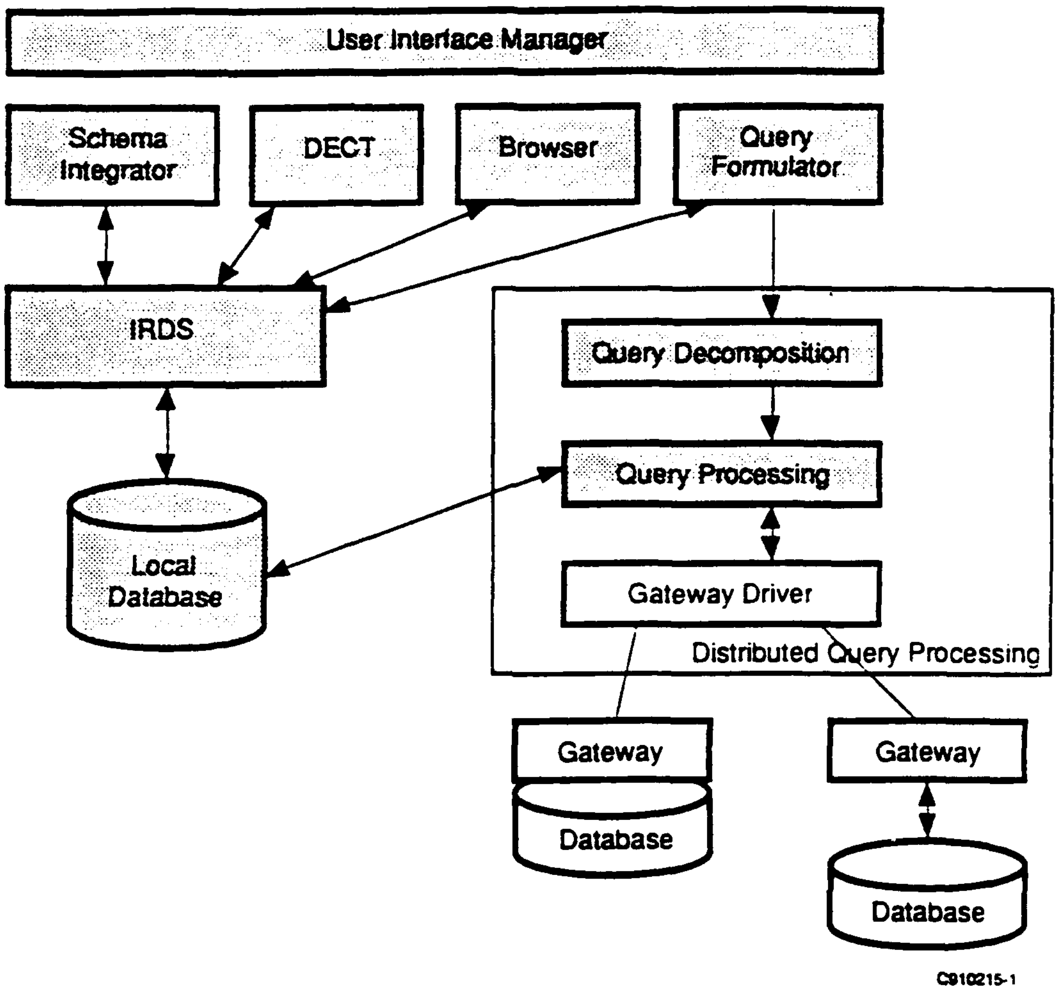


Figure 1-1. ANSWER Architecture

also controls the tool interfaces, managing input and output for each tool as well as window managing functions and graphical display functions required by the tools.

The query formulator supports the formulation of syntactically correct SQL queries. Queries may be created by filling in textual templates or by using a mouse to select items from the graphic schema display or some combination of those two modes. An SQL parser allows the graphic display of SQL queries. A graphic condition expression editor parses the SQL "Where" clause and allows the structure of the clause to be displayed as a tree. The internal structure of complex boolean conditions can be effectively edited by interacting with a graphic display of their structure. The query formulator also allows suppression of the display of subclauses within a query as the query is being built. This allows the user to focus attention on only a portion of the query at any one time. The query formulation tool produces queries executable by an SQL processor.

For queries stated against integrated schemas or stated using standard data elements, additional processing is necessary to obtain an executable SQL query. In Phase IIIA, we investigated several alternative query processing algorithms. We have implemented two algorithms for query processing that address the issues of query decomposition, execution and result composition for queries against the integrated schema. One algorithm is designed for use with standard data element queries and one is designed for use with standard integrated schema queries.

At the end of Phase IIIA, we demonstrated the use of all these tools to create, edit and integrate schemas and to create and execute queries stated against integrated schemas. The query processing system decomposed the queries into subqueries executable against individual relational databases running locally in the ANSWER environment.

Further information about these tools may be found in the Phase I, II, and IIIA final reports.

1.2 Phase IIIB

In Phase IIIB, we have further investigated query decomposition issues, including issues associated with approximate query processing and approximate relational algebras. We also investigated the security issues associated with an ANSWER-type system and identified security concerns of particular importance in this environment. We have designed and implemented gateways to two relational systems using commercial gateway products. We have also evaluated the applicability of ANSWER to an ongoing program under PO-ISM (Army Project Office–Installation Support Modules) called ILAPS (Installation Level Acquisition of Products and Services). The ILAPS application provides an ideal environment in which to evaluate the usefulness of concepts explored in ANSWER. Based on our analysis of the ILAPS application, we have developed some sample schemas that represent the kinds of information exchanged within ILAPS and have created an example to show how the schema integrator could be used to create a single integrated view of ILAPS applications. We also discuss how the ANSWER system can provide support for integrated access to ILAPS data.

Figure 1-2 shows the elements of the overall ANSWER architecture that were completed during Phase IIIB.

The specific subtasks of Phase IIIB are defined below:

- Subtask 3.1. Implement Browser/UIMS Enhancements—Port the ANSWER tool set to an appropriate hardware platform for execution with the identified distributed query processing system. In this subtask, we ported the existing ANSWER software from Sun OS 3.5 to Sun OS 4.03. We also enhanced the existing IRDS implementation to improve the usability and maintainability of the system.

- Subtask 3.2b. Query Formulation Implementation—Expand the query decomposition and query processing algorithms implemented in Phase IIIA. This will include a limited implementation of an ANSWER/RDA-like interface. The ANSWER query formulator produces SQL expressions that will be submitted using an RDA-like protocol to DBMSs for execution in a distributed processing system. The work done under this subtask will be discussed together with Subtask

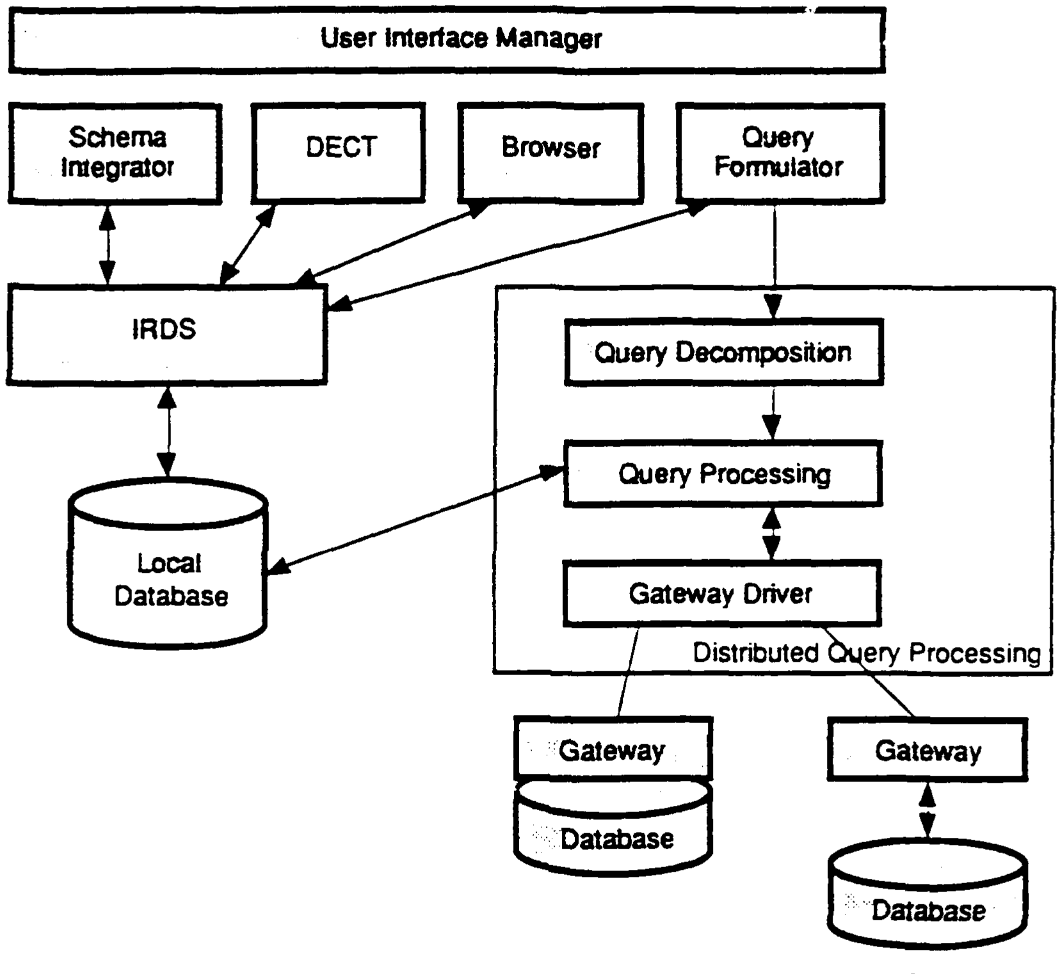


Figure 1-2. ANSWER Phase IIIB

3.6b, Distributed Query Processing. Under this task, we designed and implemented modifications to the existing query processing algorithm to use the gateways implemented under Subtask 3.6b to support access of remote systems for database query execution.

- Subtask 3.3b. AI Techniques—Under this task, we evaluated approaches to approximate query processing using approximate relational algebras.

- Subtask 3.4. Security Study—We will develop an overall approach to security issues within ANSWER and make recommendations for exploration of key concepts. We will also explore issues associated with fault tolerance of the ANSWER system.

- Subtask 3.4b Demonstration and Training—The ANSWER tools will be demonstrated against a set of sample relational schemas representing the ILAPS environment based on information obtained from Pm-ISM. The demonstration accesses a remote Oracle and a remote Informix server using SQL*NET and INET as the bases for the gateways. We developed schemas that mirror some of the information used in the ILAPS and show how the ANSWER system may be used to support ILAPS-like applications.

- Subtask 3.6b. Distributed Query Processing—The results of this tasks will be discussed together with Subtask 3.2b. We designed and implemented local gateways to support access of the Oracle and Informix data systems. We discuss the relationship of our gateway designs to the requirements for an RDA protocol.

The results of each task are described in more detail in the remainder of this report.

Section 2
Browser/UIMS Enhancements

This section addresses several issues associated with the ANSWER software:

- Improvement of the IRDS/ANSWER interface,
- Porting of the ANSWER code to the Sun OS 4.03 environment.

These tasks are intended to improve the usability and testability of the code as well as to update the environment in which the code runs.

2.1 IRDS Modifications

The IRDS interface in the ANSWER Phase IIIA software consisted of system-dependent specific file name references, both to invoke IRDS functions and to collect IRDS error messages. We modified the existing ANSWER code and IRDS code and set up UNIX environment variables to increase the flexibility of using the IRDS prototype. These modifications provided the following benefits:

1. IRDS functions may now be called from any directory. Previously, to call IRDS a specific pathname had to be used. This created problems in debugging the code when porting from one machine to another. Changing the specific file pathname references to UNIX environment variable references greatly enhanced the usability of the code.

2. Installation of the system on different machines with different directory file structures is now simplified. Prior to the simplification, any modification to the environment that changed where IRDS was placed in a directory structure required changes to the ANSWER code itself (and the corresponding recompilation of the code).

2.2 Port to Sun OS 4.03

Porting to Sun OS 4.03 required the porting of both LISP and C software as well as the purchase of upgrades for Oracle and Common LISP/Common Windows:

- Oracle Upgrade—Oracle 6.0 is required to run on Sun OS 4.03. Previous versions of Oracle do not run under the new operating system.

- Common LISP and Common Windows—An upgrade for Common LISP/Common Windows was also required.

Porting of the ANSWER code required the following steps:

1. Recompiling IRDS to run with the new version of Oracle;
2. Modifying the ANSWER calls to Oracle to be consistent with Oracle version 6;
3. Recompiling the LISP code to run with the new version of Common LISP/Common Windows. This required some changes to our handling of windows.

The porting is completed and tested. We had to acquire some patches from Franz, Inc., for their implementation of Common LISP. Those patches have been installed as part of the system. We also modified the Oracle installation to improve the performance of IRDS. We were unable to improve performance by modifying the buffer size, essentially making IRDS operate on an in-memory database. We now suspect that the performance problems are caused by the fact that we log into Oracle for each transaction. This can be changed but will require some code modifications.

Section 3
Security and Fault Tolerance for ANSWER Systems

We studied the security issues in ANSWER systems, particularly those that are unique to an ANSWER environment. This section defines the basic ANSWER environment characteristics and discusses how security considerations apply to that environment. We also briefly discuss whether or not any special considerations are necessary to support transaction commit and recovery and concurrency control.

Secure DBMSs and criteria for evaluating the security level of secure databases are discussed extensively in [TDBI90]. That report assumes control over the design and development of the complete DBMS and operating system platform. With an ANSWER system, however, no such control is possible over the operating system and the design of the individual source databases. The questions then become how much security can be enforced in such an environment and what security problems are unique to such an environment.

3.1 Definition of ANSWER Environment

The ANSWER environment is designed to include a central processor of requests for data and schema definition information. The central processor is connected via gateways to multiple DBMS systems running on multiple hardware platforms. The gateway and the central processor share a common operating system environment. ANSWER accepts a query for data from a user of the ANSWER central process or a request for schema information to browse. For schema information, ANSWER immediately initiates requests against any of the registered DBMSs. For queries, ANSWER first analyzes the query and decomposes it into subqueries to be executed against registered DBMSs. The results of individual subqueries are stored in ANSWER's own local database for recomposition before preparing the final result for presentation to the user. For standard ANSWER implementations, the individual databases typically will not have any secure data classifications. A diagram of the abstract ANSWER environment is shown in Figure 3-1.

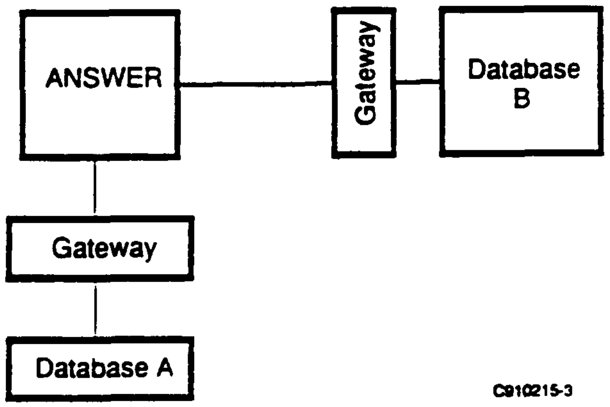


Figure 3-1. ANSWER Environment

3.1.1 How Does This Environment Differ from Others?

Several characteristics are important to keep in mind:

1. Although none of the data in the individual databases may be secure, the combination of data resulting from queries may be secure due to aggregation problems.

2. The individual DBMSs will have only access control mechanisms designed for a centralized environment. Typical security measures such as physical access restrictions may not be feasible when the DBMS is registered within an ANSWER system.

3. Security issues will arise in the creation of integrated schemas. Some examples are:

   - What is the correct integration of two schemas with different security classifications?

   - What should be done to identify and prevent schema integration that compromises security classifications?

   - How will query processing algorithms be affected by security issues?

4. Security policies may vary across sites and may be in conflict across sites. An administrative problem is determining who should be the global security administrator and how far their authority should extend into maintaining the security of individual systems. Another problem is that "need to know" may not be consistently defined across organizations.

5. While individual systems may not be managing data with multiple security levels, the ability of one user to access more than one system will almost certainly create a multilevel secure system due to the effect of aggregation of data. As a user is allowed to access more data, the security level of the data tends to rise.

These issues will be considered in more detail in the next section, where four general categories of security issues will be defined and their impact on the ANSWER system will be discussed. In addition, there are security issues associated with distributed systems in general that apply to ANSWER as well as to other systems. These issues will be mentioned as well.

3.2 Standard Security Issues

The following major categories of security issues will be discussed:

- Security Policy—definition of what security threats are to be prevented, and access control policies that will be used to enforce the policy.
- Mandatory Access Control—providing and enforcing a labeling system for data that is maintained for browsing, querying and updating of data.
- Discretionary Access Control—specifying access privileges for users and data types; definition of operations available at the user level to support access control mechanisms.
- Accountability—mechanisms to support identification and authentication of individuals accessing the data; auditing of access to determine if unauthorized access patterns are occurring.
- Assurances—verification of specified policies and enforcement mechanisms for operational system components (e.g., system architecture, covert channel analysis, facility management, system integrity and recovery); providing DBMS and operating system technology to support the specified discretionary and mandatory access controls.

We will focus on the security policy and access control policies and assurance mechanisms, particularly as they relate to ANSWER-type systems. The accountability and assurance verification issues are similar for any type of distributed system.

3.2.1 Security Policy

The major issues for security policy are to identify what threats are to be prevented and to establish policies that if properly enforced will safeguard against those threats. The policy must define permitted flows of information. For this discussion, we will assume that there are two systems, A and B, and a single central ANSWER system. Users U1 and U2 may query systems A and B through the facilities provided by ANSWER, as shown shown in Figure 3-2.

Users U1 and U2 must be allowed to view only that information in A and B for which they have access privileges. This can be managed by different methods of access control.

Other types of information flow that may need to be prevented include:

- Signaling—U1's access to information in systems A and B may provide information to some unauthorized external user X, if X has access to the sequence or frequency of queries made by U1 against systems A and B.

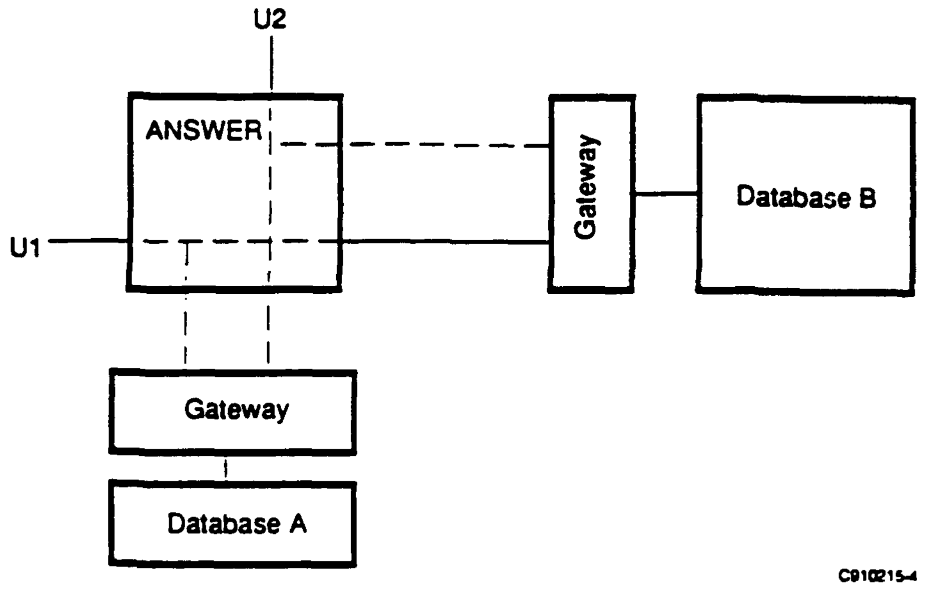


Figure 3-2. Information Flows in an ANSWER System

- Inference—U1 may have different levels of access to system A and system B. U1 may have inferential access to information contained in system B without having access to data items in B. If U1 queries system B and then based on some results from system B queries system A, that user may be able to infer some information about data items in B that he/she is not authorized to see.

Other information threats may be characterized as well. Whatever threats are specified in the security policy must be prevented through:

- Discretionary access control,
- Mandatory access control.

Each of these issues will be discussed in the context of an ANSWER system.

3.2.2 Discretionary Access Control

The primary goal of discretionary access control is to provide a means of separating users from data. This can be done by defining classes of users and classes of data to which those users have access. Discretionary access control also provides operations and objects appropriate to the enforcement of the designated user and data access classes.

Examples of appropriate operations and objects are the authorization commands provided by commercial DBMSs for authorizing users to access only specific tables or specific views of tables. In effect, this allows authorization at the level of individual attributes or rows of a table.

3.2.3 Mandatory Access Control and Assurance Mechanisms

The primary goal of mandatory access control is to provide and enforce a labeling system that will achieve the goals of the security policy. This involves specifying means to support the following DBMS operations while preserving the security labeling of data items:

- Browsing,
- Querying,
- Updating,
- Auditing,
- Integrity control.

Many of these operations cannot be successfully controlled without adequate services from the operating system. In the discussion that follows, specific concerns about DBMS and operating system dependencies will also be noted.

Browsing of ANSWER data provides some important issues for mandatory access control. It is important to specify the effect of data dictionary browsing on the security of the specific data items defined in that dictionary. Providing too much access to dictionary information may create opportunities for security compromise through inference. Jensen [JENS88] provides a thorough discussion of security issues for data dictionary information.

The data dictionary itself must also be treated as a secure object. Data dictionaries specify data validation and data indexing schemes that can provide significant information for security attacks.

Mandatory access controls must specify requirements for querying and updating that support only allowable access to data by users. There are a number of issues in providing secure query and update access, particularly in a multilevel secure environment. As noted earlier, although the individual systems in an ANSWER system, e.g., A and B, may not impose multilevel security or may not be secure systems at all, providing combined access to A and B will almost certainly create a multilevel secure system.

In addition to multilevel security, labels must be available at the correct level of granularity. There is the potential for conflict with the levels supported by the operating system, which may only provide security labeling at the level of files or directories. Frequently, mandatory access control policies require stating access control at the level of individual relations or attribute types. A general issue for mandatory access control is identifying the right set of labels and the logical specification for downgrading or upgrading security classifications.

Specifying policies for integrity control will require some approach to dealing with polyinstantiation [DeLS87]. Polyinstantiation is the creation of more than one tuple having the same primary key. This may happen when a user downgrades information from secret to unclassified. The secret tuple may have unclassified portions that the unclassified user sees. An update of the secret tuple will result in the creation of a new tuple at the unclassified level for the updated attributes that are not secret. Polyinstantiation may also occur when a user at a lower security level creates a tuple that already exists at a higher security level.

The interaction between the DBMS and the operating system must be carefully considered when establishing approaches to implement discretionary and mandatory access controls. These dependencies make it difficult if not impossible to develop a secure DBMS on an unsecure operating system [HeHW87]. Some examples of these dependencies include:

- File management,
- Auditing,
- Network services.

Most DBMSs eventually use the system file management facilities to store data. The system access control facilities can be used to support mandatory and discretionary access control at the level of individual files or directories. Smaller granularity of control, i.e., individual relations or attributes, are not typically controlled by operating system services.

Auditing services supplied by the operating system typically do not support the level of information needed for multilevel secure DBMSs. The operating system may indicate that a user accessed a database, but that information is not sufficient to provide security support. It is also important to log the user's level of access, the level of access of the data, and possibly even before and after maps of the data accessed by the user.

Network services must be trusted if the overall system is to be trusted. Information security cannot be maintained if the network is not secure.

3.3 Security Summary

Many security concerns apply to any distributed system, including an ANSWER system. These include:

- Encryption issues (both data and address);
- Identification and prevention of covert channels;
- Approaches to traffic analysis;
- Network security issues;
- DBMS and operating system interdependency issues.

Some concerns are specific to ANSWER-like systems. ANSWER systems may have security requirements, even if the source databases are unclassified due to aggregation. In addition, an ANSWER system will require multilevel security. Security policies may be more difficult to state for ANSWER systems than for other distributed systems. There are potential conflicts in the definition of "need to know" across organizations. There will need to be a global security administrator. Snapshots rather than direct system access to source systems may be required because those source systems may not have adequate security assurance mechanisms. Finally, approaches to schema integration and query processing must be modified to take into account security concerns.

A secure ANSWER system is not possible without using a trusted network and trusted operating systems as the platform to implement all the source DBMSs and the central ANSWER processor. One way to approach this is to use snapshots of the source DBMSs instead of allowing direct access to the source DBMSs. The snapshots may then be implemented on a secure platform together with the central ANSWER processor. Although this still does not address all of the issues related to security policy, schema integration and query management, it does make it possible to begin to address those issues.

3.4 Concurrency Control and Transaction Management

The basic question to be considered here is whether or not there are any special concerns with regard to concurrency control or transaction management in a heterogeneous environment such as that envisioned for ANSWER. Pu [PU88] and Gligor and Popescu-Zeletin [GLIG85] provide good discussions of these issues.

Distributed transaction management requires a central controller and subordinate transactions. Each subordinate transaction maintains its own local information to support the transaction. The global transaction management system must maintain a log of transactions recording subtransactions, parent transactions and the transaction state. In the event that a transaction is active when the system crashes, the parent restarts the transaction. Subtransactions simply wait for notice from the parent. If the transaction is in a prepared state when the system crashes (i.e., waiting for confirmation from a parent that a transaction should be committed), the child transaction either aborts the transaction or the parent aborts or redoes the transaction if the parent sent notice that the transaction was to be committed. Transaction management is essentially no different for heterogeneous distributed systems than it is for homogeneous systems. Gligor and Popescu-Zeletin [GLIG85] specify five conditions that should be met by any concurrency control mechanism for heterogeneous systems: (1) local concurrency control must provide local synchronization atomicity; (2) all local concurrency control must preserve the globally specified execution order; (3) each site should run only one subtransaction; (4) the global transaction manager must be able to identify all objects referenced by the any subtransactions; (5) an effective approach to global deadlock detection is needed. Pu [PU88] notes that good approaches for global deadlock detection are still under research.

Section 4
AI Techniques

One of the most important problems with query processing in heterogeneous database systems is that of providing answers to queries in the face of incomplete information. Integration of multiple database schemas can create many situations where incomplete information must be addressed. Some examples of how this can occur are:

1. The schemas do not match at the attribute domain level.

2. The schemas do not match at the entity or relation levels.

3. The schemas maintain the same information at different levels of granularity.

4. Schemas do not directly represent all possible subtypes that may be defined against a given universe of individuals.

Each of these will be discussed in more detail below.

4.1 Schemas Do Not Match at the Attribute Domain Level

An example of this situation is that one schema, S1, may maintain information about vendor location in the form of specific city and street address, while another schema, S2, may maintain information about vendor location only as a particular city and area within the city. The distinction between the schemas is in the domains used to represent location. The problem is that any information in S1 related to vendors in a specific neighborhood is also true of vendors for which only specific street address information is available. Techniques for dealing with incomplete information of this type will support information to be retrieved about vendors by neighborhood for data from S2 as well as S1. Similarly, there may be information associated with vendors having a specific street address that may also be true of vendors in the same neighborhood as the street address. Without some techniques for approximate query processing, there is no way to provide such "possible" information.

4.2 Schemas Do Not Match at the Entity or Relation Levels

Generally, S1 may have entities or relationships that S2 does not have. Retrieval of information from S1 or S2 across hierarchies will typically create situations where information is not union compatible. For example, if S1 has information only about noncommissioned and civilian personnel and S2 has information about commissioned personnel as well, the integration of S1 and S2 will result in a supertype Employee with three subtypes: commissioned, noncommissioned and civilian. Furthermore, the integration might also result in the creation of another type, Military, of which commissioned and noncommissioned are subtypes. Queries that request information about all personnel will have to deal with missing information in cases where the requested information is in only one of the schemas, e.g., S1. In such a case, the personnel subtypes in S2 may be included as part of the answer, but there will only be incomplete answers for queries referencing S1 information.

4.3 Schemas Maintain the Same Information at Different Levels of Granularity

An example of this situation is that S1 may contain information about the contents of a ship at the level of individual items, while S2 may contain information about ship contents at the level of type of item and quantity. If S1 represents purchase request information and S2 represents shipping information, it may be very difficult to support queries against the shipping status of particular items because of incomplete information. The best that can be done in such cases is to provide approximate or incomplete answers.

4.4 Schemas Do Not Directly Represent All Subtypes Defined Against a Given Universe of Individuals

In the case of S1 and S2, where S1 has information about noncommissioned officers among civilian personnel while S2 has information about commissioned officers, it may be important to ask about individuals who were commissioned officers at one time and are now civilian employees. Approximate techniques may be employed to compute the set of individuals who satisfy that description.

Other cases exist where approximate query processing is important to adequately deal with requests against heterogeneous systems.

In Phase IIIA, we investigated one particular approach to approximate query processing [BUDW89]. The basic ideas behind the algorithm are as follows:

1. Select rules relevant to a query.

2. Create a set of DEFINITE answers by computing an APPROXIMATE UNION of schema terms on the left-hand side of the rules.

3. Create a set of MAYBE answers by computing the JOIN of all schema terms named in the right-hand side of the relevant rules, RHS, and compute RHS MINUS DEFINITE.

4. Improve the DEFINITE answers with additional information from the RHS tables by joining tuples with identical key values.

5. Identify ANOMALOUS tuples as tuples that are in the DEFINITE answers but that are not in the MAYBE answers.

The algorithm is an effective approach to computing answers for queries about elements that are not strictly part of the defined schema structure. The approach does not extend beyond queries within individual type hierarchies. This means that it is not general enough to support arbitrary queries against integrated schemas.

As part of Phase IIIB, we have investigated other approaches to approximate query processing, [BISK83; CODD79; DEMI89; LISU90; SMLI89]. Some of these approaches define extended relational operations to manage definite and indefinite tuples within a relational system. Indefinite tuples correspond to disjunctive tuples within a logic-based system [MINK82]. The basic approach is to divide a relation into sets of definite and indefinite tuples and then define extensions to the standard relational algebra operators (Select, Project, Cartesian Product, Difference, Union) that will support the manipulation of tuples within the definite and indefinite parts of a relation. A sample definition taken from Smith and Liu [SMLI89] is the definition for approximate union. Let D and P be subsets of a relation R, where D is the set of definite tuples and and P is the set of possible tuples. The Union R[u] of R[1] and R[2] is defined as:

$$D[u] = D[1] \cup D[2]$$

$$P[u] = (P[1] \cup P[2]) - D[u]$$

That is, the union of R[1] and R[2] is the union of all definite tuples in R[1] and R[2], D[u] and the union of all possible tuples from R[1] and R[2] minus the definite tuples of R[1] and R[2].

Such approaches typically assume that the tuples being manipulated share a common set of attributes and no operations are attempted on attributes that are semantically similar but do not share the same domains.

Other work has been done on the problems of partial values for attributes and the problems of trying to manipulate tuples that share semantically similar attributes whose domains do not exactly overlap. DeMichiel [DEMI89] defined an extended relational algebra to deal with such cases. It assumes a set of rules to map attribute domains to a common domain for attributes that should be treated as semantically related (e.g., such as vendor street address and vendor city area location). DeMichiel [DEMI89] defines operations based on the standard relational algebra that combine tuples with mapped attributes, preserving the intended semantics of the attribute mapping. An example of the union operation as defined in [DEMI89] is given below.

Let the source relations be R(XZ) and S(XZ), where X is the set of key attributes and Z is the set of non-key attributes of the relations. The relation $T = R \cup S$ is:

{t|u ((t in R & (Not EXISTS u) (u in S & (u.X = t.X)))

or (t in S & (Not EXISTS u) (u in R & (u.X = t.X)))

```
or (EXISTS u)(EXISTS v) ( u in R & v in S
        & ( t.X = u.X = v.X)
        & (ALL C) ((C in Z  =>
                (t.C = u.C INTERSECT v.C))
                & (NOT (NULL t.C))))) }
```

This definition says that a tuple is in t only if the tuple is in R but not S, or in S but not R, or there is a pair of tuples u from R and v from S with the same key attributes as t, X, and all non-key attributes of t come from either u or v.

This definition is similar to the definition for union presented earlier if we consider the set P of possible tuples to be treated the same as the set of tuples with partial values. A tuple, with or without partial values, is in the result of the union in both definitions if it is definitely from either source relation (R or S). A tuple is in the possible set (from the first definition) if it is in the possible set of either source relation. A tuple is in the result of the union and has partial values (based on [DEMI89]) only if its key occurs in a tuple in both R and S and some of its values occur in both R and S.

DeMichiel [DEMI89] also extends the definitions to cover relations with indefinite tuples. The union definition is similar except that tuple status is taken into account (as it is in the first union definition above). The result of R $\cup$ S where R and S may have indefinite tuples is the same as the previous union definition [DEMI89] with one addition: some t is in the result of the union if there is a pair of tuples u from R and v from S with the same key attributes as t, X, and all non-key attributes of t come from *both* u *and* v. This extension makes the [DEMI89] definition of union essentially the same as the first definition of union.

This approach is important because it provides a well-defined means to deal with partial values not provided by the other approaches. The approach has great potential for systems such as ANSWER because there will be many cases where attributes are semantically similar but do not overlap exactly.

There is one drawback to this approach. The extended operators are defined only for union-compatible and consistent relations. Union compatibility is not a problem because relations can be made union compatible by applying the attribute mapping rules, which coerce selected attributes to share domains. The requirement of consistency is more difficult because frequently the results of operations will be inconsistent in the following sense for union:

```
(EXIST u) (EXIST v)( u in R & v in S & (u.X = v.X)
        & (EXIST C)(C in Z & (u.C INTERSECT v.C = NULL)))
```

That is, two relations are inconsistent if tuples that share the same key in the two relations do not have overlapping non-key attribute values. This means that two tuples may be inconsistent if they both have attributes with partial values where the partial values do not overlap for given attributes. This might happen fairly often in a heterogeneous system, depending on both the available attribute mappings and the external context in which the databases are intended to be used.

Two other issues must be considered in evaluating the applicability of an extended algebra approach as outlined in DeMichiel [DEMI89] to an ANSWER environment:

1. Practical implementation of domain mapping,
2. Requirement for sharing common keys.

The implementation of the domain-mapping approach essentially requires all possible attribute values to be inserted as the value of an attribute when a domain-mapping rule applies. For example, if a relation R maintains information about vendor street addresses and S maintains only information about city area, when a domain-mapping rule has been applied to S, it will have a street address attribute with a fully unspecified partial value (or all possible values from the street address domain) as the attribute value. Managing operations against such attribute values may be expensive for large domains. In addition, arbitrary subsets of domain attribute values must be represented and manipulated for individual tuples. The reason is that the results of various operations reduce the cardinality of the attribute value from the entire set of values into the domain to some possible intersection or union of possible values with partial attributes from other tuples. While this is a practical concern, it does not invalidate the applicability of the approach in general.

Another problem may present more serious hurdles. The definitions of the extended operations all depend heavily on the ability to identify common (presumably nonpartial valued) keys across relations. This was also a requirement for the Buneman and Davidson [BuDW89] approach. This requirement may not be practically met many times for heterogeneous systems without some addition to or redesign of the system. Problems with selecting good identifiers or keys are highly application and situation dependent; thus, even semantically related databases may manipulate tuples that do not have compatible identifiers. The DeMichiel [DEMI89] approach does not define operations in the presence of partial values for tuples. This issue must be addressed before this approach can be practically applied to an ANSWER environment.

Section 5
Demonstration System and ILAPS Scenarios

In this section, we describe the configuration of the demonstration system for Phase IIIB and the relationship of ANSWER to problems being addressed by projects like ILAPS (Installation Level Acquisition of Products and Services).

The demonstration system for Phase IIIB includes all software from the previous phases plus gateways to two relational DBMSs running on a different node from the ANSWER software. Figure 5-1 outlines the major components of the demonstration system.

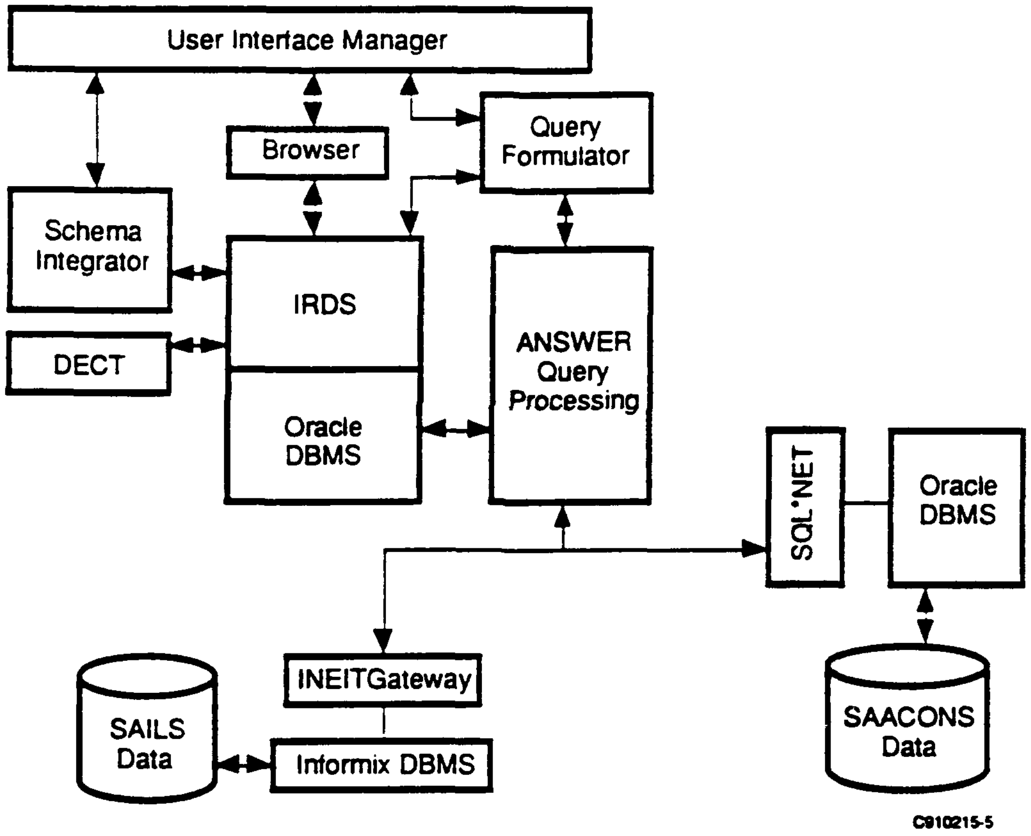


Figure 5-1. ANSWER Phase IIIB Demonstration System

We have created sample schemas that correspond to portions of the SAACONS and SAILS schemas being investigated as part of the ILAPS project. SAACONS is a contracting database and SAILS is a supply system.

The entity-relationship representations of the schemas we are using for SAACONS and SAILS are given in Figures 5-2 and 5-3. The result of integrating the two schemas using the ANSWER schema integrator is shown in Figure 5-4. The essential points to notice about the integrated schema are:

- Request category is not present in either source schema,
- Receipt category is not present in either source schema.

The integrated schema can be used to support queries against schemas.

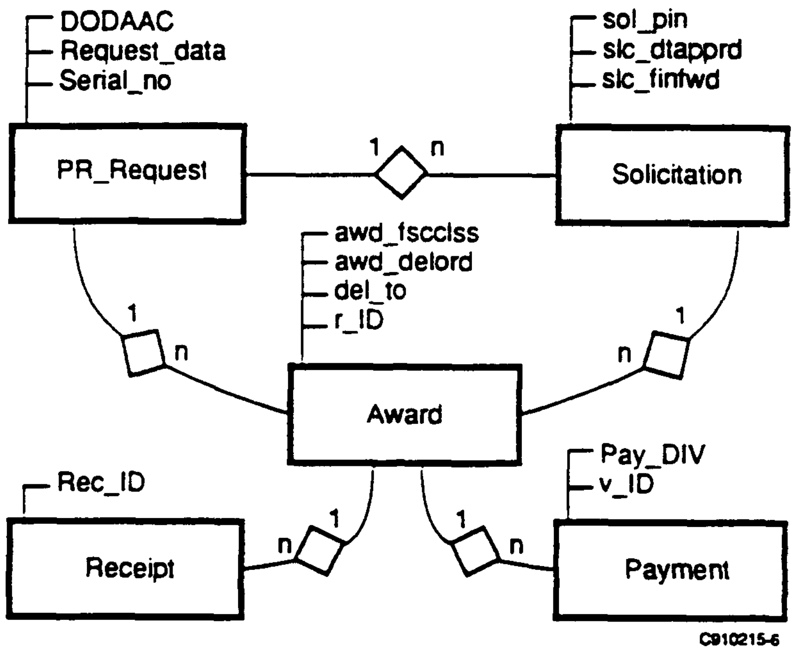


Figure 5-2. SAACONS Schema

5.1 Relationship of ANSWER Approach to ILAPS

ILAPS is intended to provide support for data exchange between supply data systems, contracting data systems and finance data systems. The ILAPS approach is to provide definitions of routes that information must take between systems and to support automatic routing of that information based on those route definitions. ILAPS depends on a single integrated view of the data available in all three types of systems. One version of that integrated view was created by representatives from each area (contracting, finance and supply), and that integrated view is currently helping to guide the ILAPS design process. Figure 5-5 illustrates this concept. The ILAPS approach will not support ad hoc queries against the integrated systems.

ANSWER can supply a different approach to the same problem. The sample integrated schema in Figure 5-4 can provide the basis for a solution where the routing of data for different organizational units is not directly represented in the system. Each organizational unit would use a copy of an ANSWER system to access the information needed from other systems. This would support both ad hoc retrieval of information from more than one system and uniform data exchange between

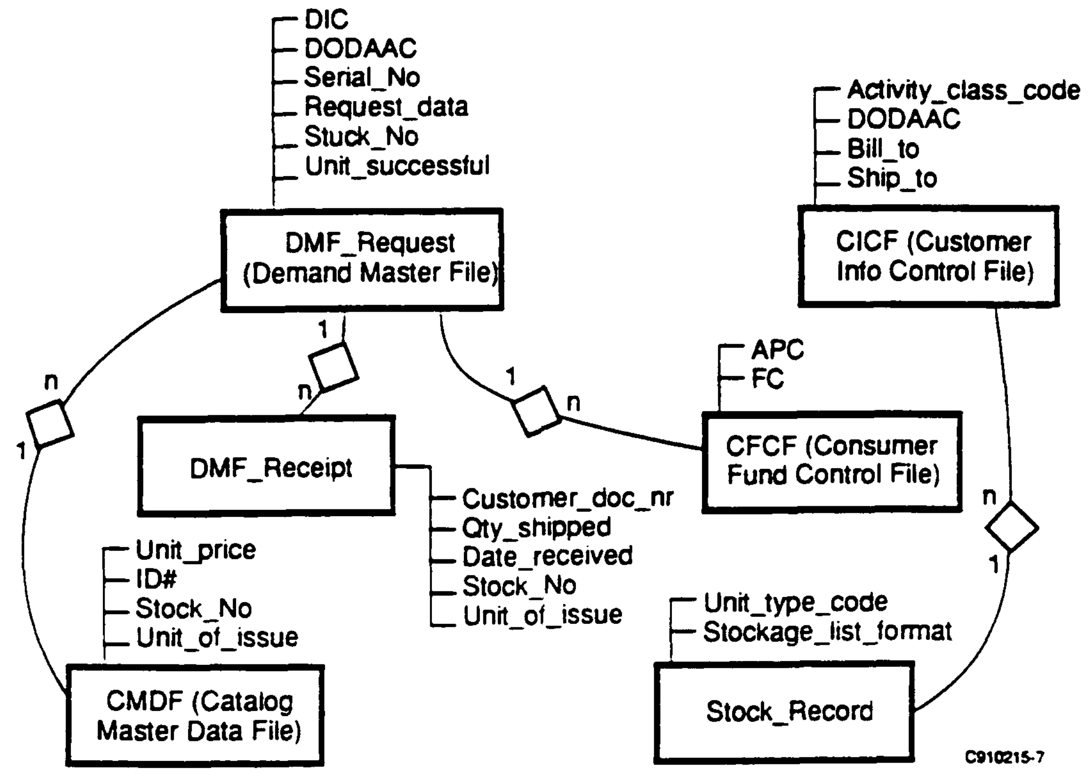


Figure 5-3. SAILS Schema

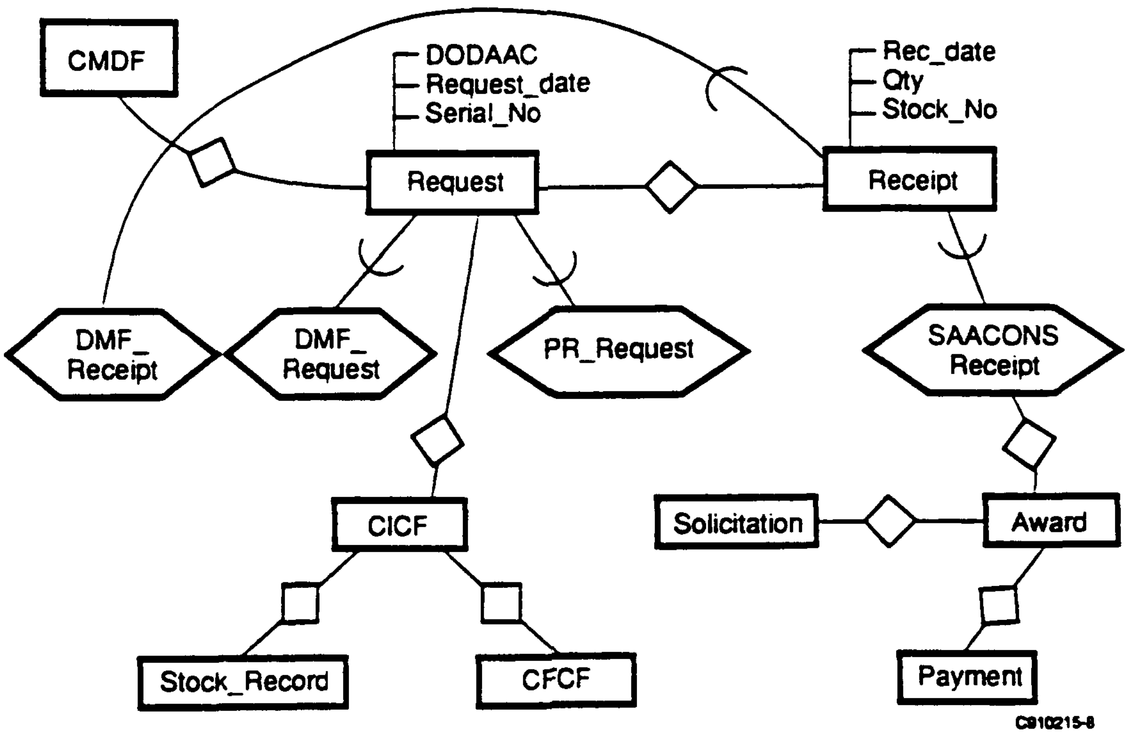


Figure 5-4. Integrated Schema

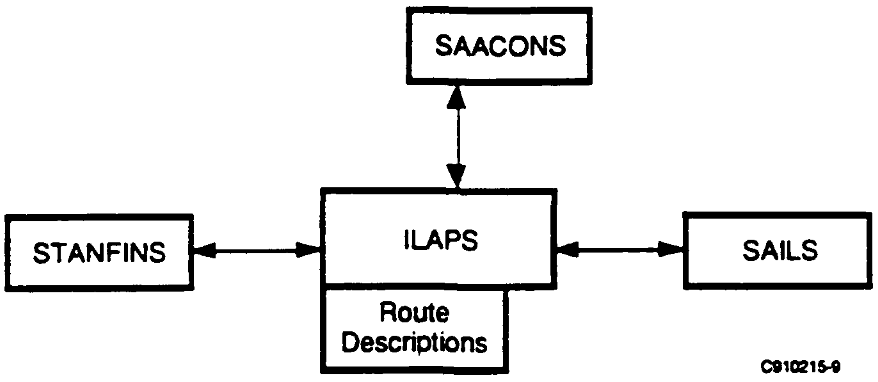


Figure 5-5. ILAPS Application

systems. Figure 5-6 shows the use of ANSWER as a means for each organizational unit to access information it requires from other systems. Each system or organizational unit can access information on an as-needed basis.

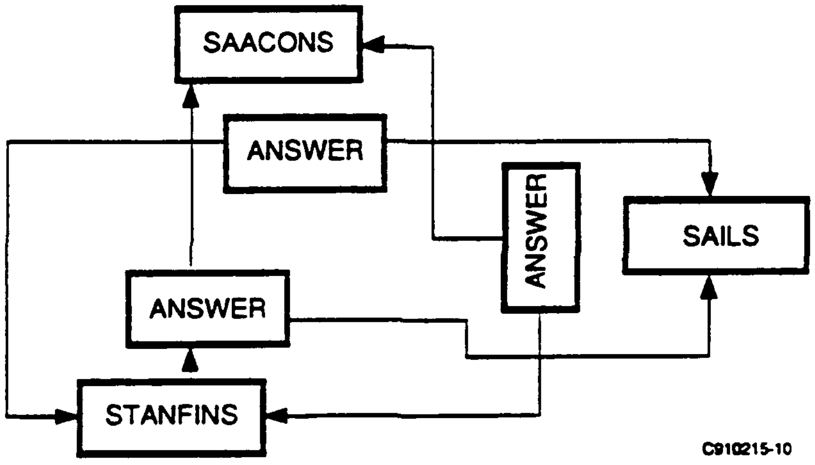


Figure 5-6. ANSWER Approach to ILAPS-Type Data Exchange

The distinction between the ANSWER approach and the approach described above for ILAPS is where the initiation of data access requests is located and the support provided for ad hoc queries. In Figure 5-6, the initiation of requests resides with the individual system or organizational unit. In Figure 5-5, the initiation of data exchange requests is controlled by the ILAPS application. Any change in a system's or organization's data exchange requirements, route or schedule of requests must be represented within the ILAPS application in Figure 5-5. In Figure 5-6, that information is not represented in any specific application. Each system or organization determines when it will initiate requests.

5.2 Demonstration Scenarios

Examples of queries that can be demonstrated with the sample schemas implemented as part of the demonstration system include the following. These queries assume that information has been initially entered into the SAILS system representing new requests.

1. Request by SAACONS system or organizational unit for all new requisitions requiring contracting attention:

   ```
   Select DIC, DOODAC, UNIT_TYPE_CODE, UI, QTY, Julian_Date,
       Fund_code, RDD
   From Demand Master File
   Where DIC = A0A & Julian date > "yesterday"
   ```

2. SAILS request for award data after contracting has completed awards process:

   ```
   Select Cage_Code, UI, qty, pr_num
   From awd_file, awd_line, vendor
   Where awd_date = "today"
   ```

3. SAACONS request for receipt data:

   ```
   Select DIC, customer_doc_number, qty_shipped, receipt_date
   From Receipt_File
   Where receipt_date = "yesterday"
   ```

These queries could be executed by SAACONS or SAILS users of an ANSWER system to request needed information that is generated by another system. As the number of systems grows, the need for a single integrated view increases in importance. The essential point is that each system or organization unit may use an ANSWER interface to access data needed from other systems. In practice, that data may actually be maintained as a separate snapshot of data by the parent system. This is necessary to prevent unexpected system loads due to processing external query requests.

The advantage of such a representation, in addition to supporting existing requirements for data exchange, is that ad hoc queries that span multiple systems can also be supported. Some examples of such queries include:

1. What items received from vendor X have no invoice yet?

2. What is the status of all requisitions being processed for vendor X?

3. How many requisitions are on hold?

4. How many requisitions have generated standard supply stockage procurement vs. local supply procurements?

Many other types of queries could be processed as well. With an ANSWER interface accessing snapshots of system data rather than entire data systems, such queries can be reliably formulated and executed. With a global integrated view of the available data, an individual needs to be an expert in the representation and storage of all data relevant to the query before the query can be formulated and executed. ANSWER will supply each user with a single integrated view of the data and manage differences in the actual statement and processing of the query.

Section 6
Distributed Query Processing

This task addresses the issues involved in supporting distributed access to the data in an ANSWER system. The problem considered here is defining how a data request issued from ANSWER is communicated to and interpreted by the target data systems and how the results are collected from the target systems and returned to ANSWER. These considerations are treated under the topic of gateway design.

The issues include:

- Command mapping,
- Data mapping,
- Directory structure,
- Gateway/data system interaction protocol.

The issues of command and data mapping are important for heterogeneous systems where the target systems do not share the same data model or query language. For Phase IIIB, we have identified the issues that must be addressed in these areas to build a gateway from ANSWER to a nonrelational system. We have prototyped a gateway from ANSWER to a relational system. In the following subsection, we will discuss the general issues and give examples of both relational-to-relational and relational-to-nonrelational data and command mapping issues. In Subsection 6.1.1, we discuss the specific approach used to implement the relational-to-relational data and command mapping.

6.1 Command Mapping

ANSWER issues data access requests to target data systems. Gateways must provide command mapping where the target data system uses a different data access language than that used by ANSWER. The current ANSWER system uses the relational query language SQL as the command language for data access requests. In cases where the target system also uses SQL there is only a minimal command mapping issue. However, many potential target systems will not use SQL as the data access language, so a mapping facility must be provided.

It is also important to note that command-mapping facilities must be provided for the correct direction of translation. For example, the command mapping specified for translating from SQL to a set of CODASYL routines will not be the same as the command mapping from a set of CODASYL routines to an SQL expression. This discussion on gateways will consider only the mapping from SQL expressions to some other target data manipulation language. The assumption is that all systems must go through the ANSWER interface to make requests to other data systems.

The possible target command languages a mapping facility must support include:

- Relational languages other than SQL,
- Hierarchical or network access through embedded applications,
- Flat file access through application programs.

6.1.1 Relational-to-Relational Mapping

Relational languages other than SQL can be translated to SQL fairly directly. Some of the standard correspondences for relational algebra are:

PROJECT: Relation-name[attribute-name] = Select attribute-name from Relation-name

SELECT: Relation-name where <boolean-condition> = Select * from Relation-name where <boolean-condition>

JOIN: Rel-name1 Join Rel-name2 = Select * from Rel-name1, Rel-name2 where Rel-name.ID = Real-name2.FK

CARTESIAN PRODUCT: Rel-name1 TIMES Rel-name2 = Select * from Rel-name1, Rel-name2

DIVIDE: Rel-name1 DIVIDEBY Rel-name = Select * from Rel-name1 where not exists Select key-2 from Rel-name2 where not exists Select key-1 from Rel-name1, Rel-name2 where key-1 <> key-2.

MINUS: Added as an operator in many commercial SQL implementations

UNION: Added as an operator in many commercial SQL implementations

The correspondences between SQL and relational calculus languages can be similarly established.

The limitations of SQL to relational algebra or relational calculus mappings center on the use of SQL ORDERBY or GROUPBY statements, and varieties of operators on attributes including SUM, COUNT, AVE, MIN, MAX, and DISTINCT. Similar constructs are not available in standard relational algebra or relational calculus languages. Each commercial implementation would have to be considered separately in constructing a command mapping. The basic approach would be to write separate application routines to support whatever function is being mapped if the function is not directly available in the target language. This approach cannot be used if the target data system does not support embedded data access routines where procedures using the target system data access language are embedded inside some application code executable against the target system.

6.1.2 Hierarchical or Network Access Through Embedded Applications

The basic approach to translating SQL statements to data accesses against target hierarchical or network systems requires a conversion of SQL statements to (typically) CODASYL programs with embedded statements for manipulating individual records with a set of records. The typical constructs in a hierarchical or network model are:

- Record,
- Attribute,
- Set of records,
- Owner of a record set,
- Member of a record set.

The basic data access operations provided by such a system are:

- Get current record,
- Find next member record,
- Find owner record within current set,
- Find member record within a set.

Mapping SQL statements to CODASYL applications requires making equivalences between series of these operations and SQL statements. A typical SQL statement such as *Select * From R1* would be expressed in a CODASYL application as:

```
Find R1 IN set S1;
Find first record within set R1-set;
while not EOF do
 { get current record
 <add attributes values to result>
 find next record within set R1-set
 }
```

A command mapping between SQL statements and CODASYL statements must specify the equivalent CODASYL templates for selected SQL templates. The relational JOIN, expressed in SQL by the statement SELECT * FROM REL1, REL2, requires two nested loops. The outer loop would read each successive instance of REL1 records. The inner loop would compare the current REL1 record with each successive REL2 record and construct rows of the joined table. If a boolean condition were part of the original SQL statement, as in SELECT * FROM R1, R2 WHERE <boolean condition>, that boolean condition would be tested in the inner loop of the CODASYL structure. Similar templates could be created for other SQL statements.

It should be noted here that translation from arbitrary CODASYL to relational expressions is much more difficult. Demo [DEMO83] describes one approach, using a dataflow analysis, that works for some cases. The general solution is still an open research problem.

6.1.3 Flat File Access Through Application Programs

The basic flat file operations are:

- Open file,
- Get record,
- Close file.

Mapping SQL statements to flat file operations requires an approach similar to that used for mapping SQL to CODASYL statements. Each SQL statement must be mapped to a template of file operations. The problem with flat file operations is that the data structures in a file may vary radically from file to file. Without knowing something about the structure of the file, it is not possible to specify the template of file operations that can be used as a mapping for an arbitrary SQL statement.

For example, consider the following sequence of characters, which might be stored in a file:

Jones,@William@S@@@@7448325.03Nelson,@Barbara@F@@@5791@91.00...

This file can be viewed in several ways:

1. A sequence of characters with no particular structure.

2. A sequence of 30-character records (no particular structure within each record).

3. A sequence of records where each record has three fields—one of length twenty, one of length four, and one of length six.

4. A sequence of records where each record has a field of type CHARACTER(20), a field of type NUMERIC(4), and a field of type NUMERIC(5,2).

The view of the file that will most easily support the mapping of SQL statements to the file operations needed to access individual records and attribute values is the view presented in 4. What is needed is a means of making a correspondence between byte streams in the file and specific record and attribute types in the SQL statement. Any of the views may be used as a starting point in making such a specification, but eventually the information about the file structure as represented by 4 must be either implicitly or explicitly part of the mapping specification.

A simple example of an SQL mapping to a series of file operations using C to access last name information might be the following:

SQL: Select lastname from EMP
File operations using C:

```
fopen(fname, r);
 while ((fscanf(fname, "%20c", &name)) !=EOF){
    /* do something with the value of name*/
    /* skip the rest of the record*/
     fscanf(fname, "%11s" );
    }
    fclose(fname);
```

It is important to note here that the mapping must be sensitive to exactly which fields are being matched because the specific arguments to the function that reads fields from the file must know how many characters are being read and what the type conversion specification (e.g., characteristic, integer, float) must be. This need to know details about the type of data being accessed distinguishes the SQL-to-file mapping from the other command mapping cases (CODASYL, other relational languages) considered in this section. Ideally, the information about data format and type and the type conversion information should be handled separately from the information about command mapping. Due to the close dependency of data type information and the specifics of file access operations for arbitrary file systems, this modularity is difficult to achieve in gateways supporting file systems.

Specific issues of data mapping and how data mapping relates to command mapping and the impact of that interrelationship on gateway design are discussed in the next subsection.

6.2 Data Mapping

Data mapping issues can be subdivided into three distinct kinds of problems:

- Data model construct mapping,
- Data type mapping,
- Schema mapping.

This subsection will focus primarily on the issues associated with data model mapping and will only briefly address the issues associated with mapping data types. In this subsection, we will also distinguish the problems addressed in schema mapping from the problems addressed in data model or data type mapping.

6.2.1 Schema Mapping Issues

Schema mapping addresses the specific representation of a specific domain and the mapping of that representation to a different representation of the same or an overlapping domain. For example, two schemas may represent information about contracting, but the schemas that are used to describe the domains may be quite different. One schema may maintain information about vendors directly with each contract, while another schema may maintain a separate representation of vendor information that is independent of information about any pending procurements or contracts. One schema may maintain codes that represent the past performance of vendors on particular contracts, while another schema may maintain no past performance information at all or may represent the information as text strings in a comments field in a vendor information record.

Some particularly difficult schema mapping issues may sometimes be seen as data type mapping issues. In such cases, the data type differences are secondary to the schema differences. For example, attribute domain mismatches, where something like vendor performance is represented as an integer code in one schema and as a character code in another schema, can create problems if the domain of vendor performance values represented by the integer codes is not structured the same as the domain of performance values represented by the character codes. The domain with integer codes may have performance values from 1 through 7, representing poorest to best performance. The domain with character codes may have five possible performance values paired with the year, representing the contract period of performance. In such cases, the fact that one schema uses integer codes and another schema uses character codes makes the codes strictly incomparable. But the more difficult problem is that because the domain mismatch is so great, the domains cannot be compared easily, making queries that range over both schemas difficult to process. DeMichiel [DEMI89] discusses such problems at length and presents an approach to deal with queries in such cases.

These kinds of differences between schemas are treated at the level of creating an integrated schema. They are not part of the problems addressed by the gateways. The gateways support only data model and data type mapping issues.

Schema mapping, data model mapping and data type mapping are independent problems. Any pair of databases may have to address one, two or various combinations of all three mapping problems. Typically, the schema mapping problems will always be present while the data model and data type mapping issues may not occur. The current discussion considers specific cases where data model mappings must be addressed.

6.2.2 Data Model Mapping

Correspondences between relational and network models are well understood and have been discussed in the literature [TSCH82]. The basic constructs needed to support data access that must be mapped between the two models are shown below:

- Relational:
  - Relation,
  - Tuple,
  - Attribute,
  - Key,
  - Foreign key;

- Network:
  - Set,
  - Record,
  - Field,
  - Member of a set,
  - Owner of a set.

The mappings between these constructs are not entirely straightforward, but they can be successfully manipulated to support interoperable queries. The correspondences are given in Table 6-1.

Table 6-1. Mapping Correspondences

Relational	Network
tuple	record
attribute	field
relation	set
key	key field
foreign key	owner-member pointer
tuple in a specific relation	member of a set
tuple in a specific relation	owner of a set

Specific mapping problems arise in the way relationships are represented. In the relational model, relationships are represented through the use of keys and foreign keys in relations. In the network model, relations are represented through the use of owner and member pointers.

Specifying a mapping between flat file data models and relational data models is possible in principal for most cases. For simple file organizations, the file is composed of a set of records, with one record on each line and record fields separated with either explicit separators or identified by a specific number of bytes.

The major issues of concern are:

- Record separators,
- Field separators,
- Multiple record structures within the same file,
- Procedurally defined file structures.

Records that are either variable in length or do not have easily identifiable record separators are difficult to map to from a relational description of the file. Similarly, fields that do not have consistent length or easily identifiable separators are more difficult to map to from a relational description.

Files that contain more than one record structure also increase the difficulty in managing a relational-to-file representation mapping. Such files may have alternating record types (i.e., a record to represent employee followed by n records to represent the employee's assignment history within the department, followed by the next employee record). Other variations on mixed file structures are possible as well. The approach to building application templates to access such files must be closely tied to the structure of the query used to access the data. In the alternating record example, the file essentially represents what would appear as a JOIN of EMPLOYEE and ASSIGNMENT records from a relational point of view. This means that to state mappings from relational accesses to file accesses, relational templates must be predefined that correspond to accesses of records within given file types.

One problem with this approach is that it is necessary to state compositional mappings. Consider the example shown in Figure 6-1. One file, EMP-ASSIGN, contains information about employees and their job assignments, and another file, DEPT-PROJECT-ASSIGN, contains information about project assignments associated with departments (e.g., each department record in the file is followed by a list of projects and assignments within projects for which the department is currently responsible).

```
EMP—Assignment               Dept—Project—Assignment
   Smith, Joh, 123456           Contracting, 129 ABC ST
        A1, special                  Thompson, 3000, 1090
        A2, standard                 A1
        A3, rush                     A6
   Jones, Harry , 782396        Finance, 853 XYZ ST
        A4, rush                     A2
        A6, rush                     A3
           .                            .
           .                            .
           .                            .
                                                  C910215-13
```

Figure 6-1. Embedded File Structure

Consider the following SQL query:

```
Select ename from EMP, ASSIGN where assignment IN (
    Select assignment from PROJ, DEPT where dname = contracting)
```

Mapping this query to the example file system requires that two separate files be accessed and the records with each file be compared. One possible procedure would be to:

1. Access all records in the department file identifying those that have dname = contracting.

2. For each of those records, compare the assignment field to the assignment field for all records in the EMP-ASSIGN file.

3. Add to the result list any enames associated with matching records.

Step 1 is equivalent to a mapping rule that could be established to define the contents of the PROJ-DEPT file. A part of step 2 is equivalent to a mapping rule describing the contents of

EMP-ASSIGN. The compositional step in mapping is the creation of a procedure that will support comparison of the selected records in the PROJ-DEPT with each record in the EMP-ASSIGN file. To support mapping from relational expressions to file access operations for such cases, a file access template must be defined for each possible join criterion that can be expressed in relational terms. This must be done for all possible pairs of files that could be compared. It is possible to define such templates in a way that supports composition of template-driven file accesses. Each template would have to be specified as a module with the following parameters:

- inputfile1, inputfile2
- <file1-field, field2-field, boolean-predicate>+
- outputfile, <field1, field2, ... fieldn>

The template specifies the names of the fields for which it was a template. The next parameter is one or more pairs of fields to be compared and the boolean predicate to be used to compare the fields. The third parameter is the name of an output file and the names (and probably type information) of the fields that will be in the output file. The output files can be used as input files for other templates. For mapping from a relational expression to a file expression, the number of templates should reflect the number of relations that can be referenced in the relational expression. If there are N possible relations that can be named in a relational expression, then N x N possible templates must be defined to support composition of file accesses.

6.2.3 Data Type Mapping

Problems of data type mapping include:

- Mapping Julian dates to other date formats;
- Mapping specialized types to more general types (e.g., money vs. real (5,2));
- Differing length character strings with the same referent;
- Quantities represented in different units of measure.

As long as data type mapping can be separated from domain mismatch or domain mapping issues (discussed above under schema mapping issues), the approach to managing data type mapping is straightforward. Such mappings must be part of any gateway design. The mapping can occur at the server gateway, with data types being converted to a common representation before returning results to the client, or data type conversion can occur within the gateway controlled by the client. The critical issue is that some type of support must be provided for data type mapping.

Commercial database systems provide some type conversion facilities as part of their export/import capabilities. Support for data mapping may require use of such facilities or more general translation facilities. Character strings frequently pose the greatest problem; for example, addresses that are represented as character strings of different lengths may require extensive abbreviation equivalence tables and/or algorithms that control the abbreviation or extension of strings to fit within an expected format.

6.3 Directory Structure

An important component of the data access gateway is the directory that is used to identify logical systems and pair those systems with specific hosts. When connecting to more than one existing commercial system, certain issues must be addressed to define a consistent directory structure. Requirements for identifying and accessing a data system will vary from one commercial system to another. Databases may be identified by a variety of techniques, including:

- Path names within a UNIX directory structure;
- User login information (uid, pwd), a specific host name, and a system id.

Directory structures within a gateway must be able to handle such variation. Directory structures also must be compatible with management of hosts within a network. In designing the gateway and directory design for this project, we have assumed that the network was a TCP-IP-based network using networked Suns. Other network configurations would have some effect on the content and use of the directory information.

6.4 Gateway/Data System Interaction Protocol

The basic approach to gateway design for ANSWER currently includes the following components:

- Connect (dbname, dbtype);
- Execute_query (query, result);
- Disconnect (dbname).

Connect uses the directory and login information required for for the given dbtype associated with the dbname to establish a connection with a particular database.

Execute_query specifies the query to be executed as well as a file in which the results of the query execution will be written. Note that the more general solution here would be to use buffers instead of files, where the buffers are sent over the network between client and server. A limitation of the existing gateway demonstration is that it requires both client and server to be able to reference a single logical file system. A more general configuration of an ANSWER system would require buffer communication instead of file communication because a common file system may not exist between designated servers and ANSWER.

Disconnect logs off the database in whatever way is required by the system.

Each data system must maintain a gateway in the form of a local application that implements these three functions with the given interface.

ANSWER maintains a gateway driver that manages the execution of queries against individual systems. Once the individual queries have been executed, the results must be loaded into the central ANSWER database so that the results can be recomposed. The general architecture is shown in Figure 6-2.

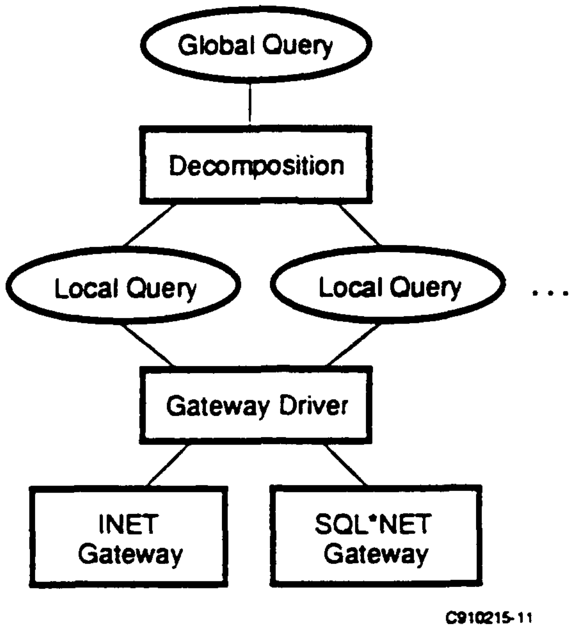


Figure 6-2. ANSWER Distributed Query Processing

This section has discussed the varieties of mapping issues that must be considered when designing gateways for heterogeneous systems. Figure 6-3 shows the same ANSWER gateway architecture annotated with the mapping concerns that are addressed by each module.

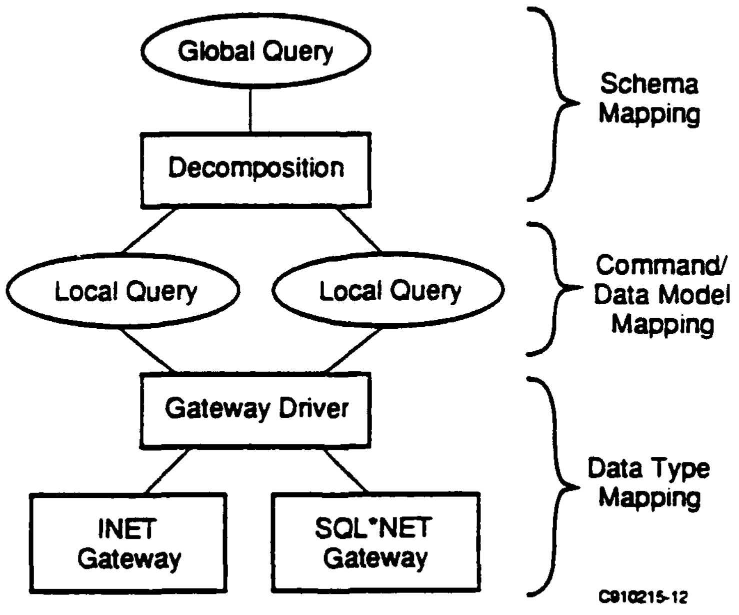


Figure 6-3. ANSWER Gateway Mapping

The query decomposition modules in ANSWER manage the schema issues.

The gateway driver accepts queries that have been mapped to local system terms and manages the connection with the local data systems and the execution of the query. It is at this point that the command and data model mapping steps would have to be supported for a heterogeneous system. The current ANSWER demonstration system only accesses relational systems using SQL, so no command or data model mapping is required.

The implementation of each individual gateway assumes that it receives a query that may be executed in the local server data access language. Each gateway retrieves the data and data type information to be sent back to the gateway driver. Thus the local gateway partially supports the data type mapping step.

The gateway driver in ANSWER loads the local query results into a single central database, doing any additional type conversion that is required.

The ANSWER query processor takes the results of individual server queries and composes them into a final result using either joins, outerjoins or unions, as required. It is at this point that the final issues with schema mapping would be resolved. This includes any issues of domain mismatch that require approximate query processing techniques.

The current ANSWER demonstration gateways are implemented using commercial products: Informix and its gateway product INET; and Oracle and its gateway product SQL*NET. The specific configuration of the demonstration system was discussed in Section 5.

6.4.1 RDA Protocol

The RDA (remote data access protocol) is an emerging ISO standard for data access against remote systems. RDA is mentioned in the GOSIP User's Guide [BOLA89] as a protocol that will be integrated with GOSIP as an association control service element (ACSE). RDA specifies a protocol for connecting with a remote system, executing a query, returning a result, and disconnecting from the remote system. RDA also specifies protocols for managing the session as a transaction. RDA is best understood as providing a collection of services where each service follows a structure of:

- Request,
- Indication,
- Response,
- Confirm.

A single service (e.g., opening a database) is defined as a client request for the service, followed by an indication that the service was invoked from the server. The server processes the request, and at a later time, the server creates a response (either a result or an error). The client creates a confirmation for the response. The RDA protocol allows the execution of services to be asynchronous so that a client may invoke multiple RDA operations without waiting for results of any previously invoked operations. Asynchronous execution creates an environment where existing resources can be used more effectively.

To support this asynchrony, RDA supports the notion of an RDA dialogue. An RDA dialogue is a cooperative relationship between an RDA client and an RDA server. If RDA were applied to ANSWER, the ANSWER gateway driver could be considered as the RDA client and the remote data systems as the RDA servers. RDA operations can only be executed in the context of an existing RDA dialogue. An RDA dialogue has a unique identifier, and the possible states of any RDA dialogue are defined by a state transition network. Each RDA service primitive is defined as causing a transition from one dialogue state to another dialogue state. This captures the dependencies between service primitives so that the correct preconditions and postconditions exist for the execution of each primitive.

The basic RDA service primitives include the following:

- Begin-Dialogue,
- End-Dialogue,
- Begin Transaction,
- Commit,
- Rollback,
- Cancel,
- Status,
- Open,
- Close,
- ExecuteDBL.

(Note that distinctions between request. confirm, invoke, and response are suppressed in this list.)

The RDA protocol is much more highly structured than the protocol implemented in the current ANSWER demonstration system. The current ANSWER system has simple equivalents to the RDA service primitives: Open, Close, ExecuteDBL. The current ANSWER system is not asynchronous. For each request made to a server, the ANSWER system must wait until a response of some sort is received before invoking another operation.

To modify the existing ANSWER system to make a more "RDA-like" protocol, it would be necessary to distinguish the four components of each operation (request, indication, response, confirm) and define and implement a state transition system at both the client and the remote servers to enforce the definitions of the ANSWER "RDA-like" gateway operations.

Complete implementation of an RDA protocol requires that the underlying system is OSI compliant. In particular, RDA is defined at the Application layer of the OSI model. RDA definitions reference the OSI concepts of ASE (application service element) and AC (application contexts).

6.5 Summary

This section has presented the general issues involved in creating a variety of heterogeneous gateways. The issues include: schema mapping; command/data model mapping; and data type mapping. Concern with communication heterogeneity and asynchronous gateway architectures can be addressed in the context of the RDA protocol for remote data access. Further details on the design of the current ANSWER gateways and information about adding new gateways to ANSWER is presented in the Code Documentation volume.

Section 7
Bibliography

[BISK83] Biskup, J. "A Foundation of Codd's Relational Maybe Operations," *ACM Transactions on Database Systems*, Vol. 8, No. 4, pp. 606–636, 1983.

[BOLA89] Boland, Tim. Government Open Systems Interconnection Profile User's Guide, NIST-SP500-163, August 1989.

[BuDW] Buneman, P., S. Davidson and A. Waters, "Federated Approximations for Heteroegeneuos Databases," *IEEE Techmical Committee on Office Automation Newsletter*, Vol. 3, No. 2, pp. 27–34, August 1989.

[CODD79] Codd, E.F. "Extending the Relational Database to Capture More Meaning," *ACM Transactions on Database Systems*, Vol. 4, No. 3, pp. 397–434, 1979.

[DeLS87] Denning, Dorothy E., Teresa Lunt, Roger Schell, Mark Heckman, William Shockley. "A Multilevel Relational Data Model," *Proc. 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 220–234, April 1987.

[DEMI89] DeMichiel, L.G. "Performing Database Operations over Mismatched Domains," PhD Dissertation, Stanford, June 1989.

[DEMO83] Demo, B. "Program Analysis for Conversion from a Navigational to a Specification Data Base Interface," *Proc. 9th International Very Large Data Bases Conference*, Florence, Italy, pp. 387–398, 1983.

[GLIG85] Gligor, V., and R. Popescu-Zeletin, "Concurrency Control Issues in Distributed Heterogeneous Database Management Systems," in F. A. Schreiber and W. Litwin (eds.), *Distributed Data Sharing Systems*, pp. 43–56, North Holland Publishing Co., 1985.

[HeHW87] Henning, Ronda R., Brian S. Hubbard, Swen Walker. "Computer Architectures, Database Security, and an Evaluation Metric," *Proc. IEEE Third International Conference on Data Engineering*, Los Angeles, CA, pp. 518–525, February 3–4, 1987.

[JENS88] Jensen, Nancy R. "Implications of Multilevel Security on the Data Dictionary of a Secure Relational DBMS," *Proc. 1988 IEEE Symposium on Security and Privacy*, Oakland CA, pp. 58–65, April 1988.

[LISU90] Liu, Ken-Chih, and R. Sunderraman. "Indefinite and Maybe Tuples in Relational Databases," *ACM Transactions on Database Systems*, Vol. 15, No. 1, pp. 1–39, March 1990.

[MINK82] Minker, J. "On Indefinite Databases and the Closed World Assumption," in *Lecture Notes in Computer Science*, N138, Springer-Verlag, New York, 1982, pp. 292–308.

[PU88] Pu, Carlton. "Superdatabases for Composition of Heterogeneous Databases," *Proc. IEEE 1988 Data Engineering Conference*, pp. 548-555, 1988.

[SMLI89] Smith, K.P., and J.W.S. Liu. "Monotonically Improving Approximate Answers to Relational Algebra Queries," *IEEE COMPSAC*, pp. 234–241, 1989.

[TDBI90] "Trusted Database Management Systems Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-021, Version 1, 1990.

[TSCH82] Tsichritzis, D., and F. Lochovsky. *Data Models*. Prentice-Hall, Englewood Cliffs, NJ, 1982.