AFRL-IF-RS-TR-2001-129
Final Technical Report
June 2001

EXTENSIBLE REQUIREMENTS MANAGEMENT ARCHITECTURE

Odyssey Research Associates

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J779

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-129 has been reviewed and is approved for publication.

Deborah A Cerino

APPROVED: DEBORAH A. CERINO
Project Engineer

James A Collins

FOR THE DIRECTOR: JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

EXTENSIBLE REQUIREMENTS MANAGEMENT ARCHITECTURE

David Rosenthal

Contractor: Odyssey Research Associates
Contract Number: F30602-00-C-0066
Effective Date of Contract: 28 March 2000
Contract Expiration Date: 28 March 2001
Short Title of Work: Extensible Requirements Management Architecture
Period of Work Covered: Mar 00 – Mar 01

Principal Investigator: David Rosenthal
Phone: (607) 257-1975
AFRL Project Engineer: Deborah A. Cerino
Phone: (315) 330-1445

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Deborah A. Cerino, AFRL/IFTD, 525 Brooks Rd, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jun 01	Final Mar 00 - Mar 01

4. TITLE AND SUBTITLE

EXTENSIBLE REQUIREMENTS MANAGEMENT ARCHITECTURE

5. FUNDING NUMBERS

C - F30602-00-C-0066
PE - 63760E
PR - IAST
TA - 00
WU - 19

6. AUTHOR(S)

David Rosenthal

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Odyssey Research Associates
Cornell Business & Technology Park
33 Thornwood Drive, Suite 500
Ithaca, NY 14850-1250

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFTD
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2001-129

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Deborah Cerino, IFTD, 315-330-1445

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The goal of this effort is to facilitate the appropriate control of security configuration parameters based on goals and available systems after a cyber attack. This report addresses a method to aid in the reconfiguration of systems after a cyber attack. Most current research on requirements focuses on how to satisfy requirements under some small set of circumstances - possibly an expected situation and/or a safe fallback position. When some systems and components have been disabled by enemy attacks, it may not be possible to meet some predetermined set of system security requirements. In this case, the cyber commander needs automated assistance in choosing an alternative that represents the best compromise between the "required" and the achievable. To do this, we have to relate system between the "required" and the achievable. To do this, we have to relate system requirements to configurable parameter settings and describe precisely how the satisfaction of low-level requirements affects the satisfaction of requirements at the system level. This will help present decision-makers with a picture of what configurations are possible and what trade-offs they involve. This work is a first step towards systematizing this process.

14. SUBJECT TERMS

security, requirements, reconfiguration

15. NUMBER OF PAGES

68

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Abstract

It can be very difficult to understand the implications of losing system resources: which requirements are no longer being met, which could be met (possibly at a reduced level) by reconfiguration, what tradeoffs reconfiguration must make. The goal of this project is to improve that understanding—in particular, to aid in reconfiguration after a cyber attack.

Sometimes an analytical or simulation model—e.g., a model of traffic in the telephone system—solves the problem by providing a good map between resources and goals. What is to be done in the absence of such a model, when a complex system has goals that are heterogeneous, changeable, and not easily described *a priori*? Most current research limits itself to a small set of loss scenarios and/or to defining a set of safe fallback configurations, which may not provide an adequate response to the unpredictable effects of malicious attacks. To choose a good compromise between the "required" and the achievable, a cyber commander may need automated support for inferring how the satisfaction of low-level requirements affects the satisfaction of requirements at the system level.

To provide that support we must relate the settings of "configuration parameters" to system goals (for both functionality and security). We propose to make that connection by systematically factoring responsibilities and requirements among system management, applications, and mission planning. We investigate the strengths and weaknesses of this strategy by applying it to examples—a field operations website, security logging on a PC, managing CORBA security services. We indicate how useful models might be constructed and sketch a generic "template" as a first step toward systematic development and tool support.

Table of Contents

List of Figures

Summary

If system resources are lost, as the result either of accident or of a deliberate attack, the consequences for overall system goals can be very difficult to understand. This joint project by Odyssey Research Associates and CoGenTex attempts to improve that understanding—in particular, to aid in reconfiguration after a cyber attack. We wish to help system administrators to determine what requirements are no longer being met and to present decision-makers with a picture of what reconfigurations are possible and what tradeoffs they involve.

Some tightly focused problems—e.g., understanding traffic in the telephone system—are already supported by analytical or simulation models capable of providing a good map between resources and goals. But many complex systems have goals that are heterogeneous, changeable, and not easily described *a priori*. Most current research applies some combination of two basic strategies: Mask faults by fault tolerant techniques for managing redundancy; define a set of safe fallback configurations that provide acceptable responses to anticipated loss scenarios. But these strategies may not always provide an adequate response to the unpredictable effects of malicious attacks.

What other options are available for more flexibly relating resources and goals? We suspect that the complexity of the problem will overwhelm both top-down and bottom-up strategies, because the distance between the concrete parameters that a system administrator can set and the system's goals is so large. So we start in the middle and work in both directions. This amounts to a way of factoring responsibilities among system management, applications, and mission planning.

We introduce mid-level abstractions of application programs that specify their "modes of operation" and the resource requirements for each mode. Typically, only a vendor or domain expert would have the knowledge to write such specifications. The coarseness of this abstraction is its virtue. One overwhelming problem is reduced to two that are merely very difficult: relating concrete configurations to the application modes they support and relating available application modes to more general system goals.

The specific question we consider is how a system administrator might adjust "security configuration parameters"—such as the kinds and amounts of audit logging—in order to compromise between requirements for security and functionality. We investigate the strengths and weaknesses of our strategy by applying it to some simple examples—a field operations website, security logging on a PC, managing CORBA security services. We make a first step toward systematizing this method in a way that could provide a basis for automated tool support.

1. Introduction

1.1 The problem

Typical strategies for operating with damaged or failed components may not respond adequately to the unpredictable damage that can result from a malicious attack. These strategies seek to mask faults (by replication, voting protocols, etc.) and/or to define degraded modes of operation that accommodate *anticipated* patterns of damage.

Recovery from a cyber attack may require a more flexible response. This report considers the specific problem of reconfiguring a system by adjusting its "configuration parameters"—that is, decisions about such things as what events will be audited and how long audit records will be kept. These parameters determine what security services are provided (therefore, what security requirements can be satisfied) and what resources those services consume. Administrators may change them in order to effect tradeoffs between security and other goals. We wish to provide an analytical framework for making those tradeoffs, one that could serve as the basis for a degree of automated support.

In the abstract, it does not matter whether an administrator is configuring a system initially or reconfiguring after resources have been lost—but we are particularly interested in reconfiguration after a cyber attack. In that case, the problem becomes acute because time is short and because unpredictable losses may mean that no pre-planned response is satisfactory. In the abstract, it does not matter what parameters are being adjusted. The fundamental problem is always the same: how to connect high-level goals with concrete parameter values.

Consider a simple example. Suppose that a system, when functioning normally, collects intrusion detection records, but that a cyber attack has degraded the available computing resources. Suppose that the Information Officer/Cyber Commander wants to reduce the resources consumed by the intrusion detection system until certain key mission objectives are met—and, of course, wants to avoid degrading security capabilities more severely than necessary.

What are the alternatives? A complete analytical solution to the problem seems out of the question: The security consequences of small cutbacks in collecting intrusion detection records depend, radically, on what attacks are actually launched: losing information that is 3 months old may allow certain attacks to proceed unnoticed, may delay the detection of others, and on others may have no impact at all. Theoretically, one could estimate the effects with a statistical model—but for a number of reasons we believe such a model is unlikely to succeed. At the other extreme from fine-tuning is the (bad) solution of simply turning off security capabilities in an *ad hoc* way. Unfortunately, this "solution" is often adopted because people making decisions do not understand their implications. There is a middle position: Find ways to help administrators make the right decisions by providing an informative "view" of their situation and, where possible, automated support for choosing the right settings. By

properly parameterizing configurations, we can support flexible, reusable analysis. Such a strategy does not preclude the use of fallback solutions, which may often be sufficient.

1.2 The approach

Our approach is conceptually simple but, as will be seen, not particularly easy to implement. We structure the description of a system around the dependencies between resources and services. This description forms the basis of a design process that recognizes multiple objectives and the complex interdependencies between them. It designs for the possibility of loss by making explicit the information necessary to understand the functioning of a damaged system. By contrast, a "standard" design process decides what resources are necessary to meet a system's goals and anticipates damage in two ways: by providing fault tolerance (where possible) to reduce the possibility of falling into a degraded state; and/or by anticipating what kinds of damage are possible and defining fallback modes to accommodate them. These standard methods are adequate for many environments. They will be strained when mission objectives and security concerns are difficult to anticipate at design time (as in a system used for a variety of heterogeneous purposes, and having constantly changing "missions") or when a cyber attack has caused unpredictable forms of damage.

Our methods factor responsibilities and requirements among applications, system management, and mission planning. We rely on the vendors of applications and services to define *modes* of operation for their products, along with the resources required and the services provided by each mode. These modes determine the granularity of the system description. We similarly rely on system administrators to indicate how underlying resources are configured (perhaps with automated support for discovering these settings). Mission specialists are responsible for assessing how well the functionality/security available meets their needs at a particular time. We will show by example how this basic data may be organized into system descriptions that help us to infer the effects of security settings on the modes in which the system may operate.

Descriptions in our style contain "slots" corresponding to clearly posed problems that may be difficult to solve. One example is the problem of specifying the properties of an aggregated or "summary" resource. Suppose that disk space is available on a variety of networked machines and that software mechanisms support its use in a distributed fashion. How do we describe "the" storage provided by these disks in ways that we can usefully relate to the needs of applications? The total capacity of the available disks is relevant, but so are network delays—and so, therefore, is the performance of the underlying network, which may itself be damaged. The question arises not only when making "normal" use of applications with constrained resources, but also when a critical situation makes it necessary to use an application "abnormally." Solving specific instances of such problems—whether by analytic methods or simulation—is outside the scope of this investigation. We can only argue that the "slots" occur in the right places, identifying important concerns.

System descriptions created by our methods are examples of structured requirements. Many existing systems for requirements managements—including EMMA and SSAT[1]—exploit this style. It should therefore be possible to incorporate our methods into COTS products, forming the basis for support tools that

- infer relations between security settings and the available modes of operation
- help relate security settings to mission objectives (not discussed in this report)
- cooperate with resource monitoring tools to respond to attacks automatically

This work is exploratory. In particular, tests that compare our approach to "standard" methods—for example, by simulation—are premature. Sections 3, 4, and 5 consider a number of simple examples. Section 6 contains a preliminary sketch of a generic model that could serve as a basis for systematic development and tool support. The possibilities for tech transfer are briefly discussed in section 7; and section 8 provides some final observations.

We begin by describing some existing tools for requirements management and some work related to our approach.

2. Background

2.1 Techniques

Maintaining a system that can operate in varying environments, and will tolerate some amount of damage, is a longstanding and much studied engineering problem. The oldest solution is to design modes of operation that can function with given sets of resources. More recently, fault tolerant techniques have been developed to mask failures by introducing and managing redundancy.

The problem we are addressing includes two significant complications, both of which limit the usefulness of design-time solutions: The kinds of damage that might arise from a cyber attack are difficult to predict. The system's mission (and, therefore, the relative importance of specific system capabilities) will vary. As a result, any comprehensive set of operating modes or fallback configurations may be prohibitively large (particularly at fine levels of granularity, such as specifying the amount of disk space one should allocate to auditing). For certain constrained problems (e.g., network recovery), specialized techniques may be available to aid reconfiguration decisions (see Section 2.2). The engineering support described here is more general, although these techniques have a natural place within our methods.

Traditional design-time techniques should be used, as far as possible, to help tolerate cyber attacks or define fallback positions, but we must also allow for reconfiguration

---

[1] EMMA and SSAT originated in some of our earlier work on requirements management. The current work goes beyond them to consider situations in which the set of available resources is much less predictable.

during operations. For example, the commander of an under-performing system may decide to limit auditing, in order to reduce the system load, but may still wish to maintain some appropriate, if lower quality, security capabilities.

No existing tools address our problem directly, but many provide services (project and requirements management) similar to those we require and many provide analysis functions that our methods could exploit directly. Section 2.3 describes some management tools of particular interest and briefly discusses the possibilities of integrating our methods with them. Section 2.2 briefly discusses the kinds of analysis functions that are relevant.

2.2 Analysis functions

Our methods often presuppose the availability of solutions to particular problems, such as estimating the storage requirements for an auditing log or the performance impact of adding or eliminating certain processes. It is our job not to develop such tools but to show how existing (or future) tools could be exploited within a general framework for helping to solve reconfiguration problems. This section organizes our needs into a few broad categories—in all of which there currently exists at least some tool support.

2.2.1 Assessing resources

The basic questions we consider ask what conclusions can be drawn "given that certain resources are available." The obvious question is: how is this information *given*? There is a wealth of monitoring tools for determining the configuration of a system, the status of network nodes, performance statistics, etc.

2.2.2 Specifications for applications

As will be seen, our methods require that vendors describe their applications in terms that allow us to relate the availability of resources to the functionality of the application. The obstacles to obtaining such specifications are at least as much practical and political as they are technical.

2.2.3 Relating resources to application needs

Whether the combination of available resources really meets an application's resource requirement can sometimes be hard to compute. E.g., will a reconfigured network meet the bandwidth needs of an application? Numerous techniques and tools have been developed to support this analysis for various specialized problems.

2.2.4 Relating missions to options

It may be difficult to understand in what sense a collection of applications operating in particular modes will meet the overall objectives of the organization. For example, what functionality and security is necessary to meet the current mission requirements and to cope with some current cyber attack. There is ongoing research on mission modeling approaches and tools that could facilitate this.

2.2.5 Reporting

To support high level decision-making tools must generate suitable reports. That problem is addressed partly by our modeling methods (which attempt to articulate information so that it can be expressed without confusing low-level detail) and partly by the technology of report generation. There is a rich supply of tools for report generation (including tools by CoGenTex).

2.3 Requirements management and project management tools

We indicate how some of our proposed methods might be integrated with some COTS products and research prototypes for project and requirements management.

2.3.1 EMMA

The Evolution Memory Management Assistant (EMMA [1]), developed by CoGenTex for the DARPA Evolutionary Design of Complex Systems program, is a meta-model and tool that provides a convenient structure to explicitly record the following types of information:

1) The properties that a system (or component) must fulfill and the context under which it is to be developed.
2) The planned sequence of releases of a system, and the information about what system properties and context elements will be available in what releases.
3) The alternative solutions considered for each system component.

These features together provide information management to support the development and evolution of a complex system. They allow for reasoning about interdependencies of subsystems and about what each member of the development team is expecting from the others. EMMA supports developers in thinking in terms of system evolution rather than the separate activities of system development and system maintenance. By explicitly representing alternatives and future releases, developers have a convenient way to record their thoughts about why they made the design decisions they made and what future circumstances would cause them to revise those decisions.

Design evolution information in EMMA is organized around five interrelated concepts:

- *Functional requirements*;
- *Context*: assumptions, resources, and design constraints under which those requirements are to be achieved;
- *Development goals,* which cluster related requirements and the associated context information;
- *Solutions*, which are software engineering approaches to meeting those goals; and
- *Evolutionary Changes,* which may alter requirements, context, goals, or solutions.

The goal of EMMA is to manage information used in software development throughout a system's life cycle. EMMA differs from other tools in several respects: (1) EMMA manages not only the system requirements that developers must satisfy, but also the context in which the developers develop the system. This contextual information includes assumptions about the operating environment and user behavior of the fielded system, as well as assumptions about externally supplied products (tools, infrastructure, etc.). (2) EMMA supports explicit planning of software evolution. Following the release-based software development model, EMMA enables developers to plan for several future versions of a system at once, so that the system can be built not only to satisfy current requirements, but also to anticipate future requirements. (3) EMMA supports communication among all the members of the development team, including project managers, software designers, and software developers. (4) EMMA relates system development tasks to technical issues, to help make sure that all members of the project are informed about the consequences of development decisions.

Like project management tools such as Microsoft Project, EMMA partitions work on a project into a number of smaller parts, and allocates each part to one or more members of the development team. Also like project management tools, EMMA keeps track of the dependencies among parts of a large project. However, in project management tools, the emphasis is on scheduling and resource allocation for tasks, while in EMMA the emphasis is on assumptions and requirements for system components. In project management tools, there is no convenient way to connect the various tasks with system components, and there is no way to automatically determine functional dependencies among tasks; these dependencies must be inserted by hand. On the other hand, EMMA keeps track of dependencies among system components in terms of the properties and assumptions declared for each component. In this way, EMMA has a more detailed understanding not only of dependencies, but also of the reasons (rationale) for dependencies. Thus, EMMA is more flexible in considering contingencies and alternative approaches to building a system. Because EMMA is not concerned with scheduling or with resource allocation, its functionality is complementary to that of a project management tool.

A drawback of trying to directly incorporate the ideas of the current effort into EMMA, is that EMMA is not a COTS product. Thus, the tech transition path with a pure EMMA approach would be difficult. An alternative is to incorporate some of the key ideas of EMMA and the new ideas of this project into COTS product.

2.3.2 RDD-100/RDD-2000/CORE

RDD-100 [2] is a comprehensive package for supporting system engineering developed by Ascent Logic Corporation. The RDD-100 system is used to manage, analyze, specify, track and record decisions, verify and document large complex enterprises and/or developments. RDD-100 contains a System Description Database that is used to maintain a consistent description of a system across a number of possible presentations. With RDD-100, systems are described using graphical behavior diagrams. These diagrams can be automatically converted into the more traditional styles of Functional Flow Block Diagrams (FFBDs) and the Integrated Definition Language (IDEFO), which

is an adaptation of the SADT notation. The behavior diagrams can be refined in a hierarchical manner and viewed in a number of ways. This information can be exported though external text files, called "rdt" files.

RDD-2000 extends the RDD-100 tool with a client-server data model, enhanced user interface, and task-oriented work model. It includes features for comparing versions, and variants on models, as well as metrics on various program aspects such as change, productivity, and documentation. It has a public API for accessing and modifying model information.

The CORE tool [3], developed by Vitech, is similar to RDD-100 in both functionality and style. It has a flatter representation of behavior than the RDD-100 behavior diagrams. (In particular, there is no separation of a system into "FNET" and "RNET" behaviors.) The latest release of CORE that was just announced, version 3.0, supports a public API.

The recent enhancements of these products with better support for third-party interaction with the underlying data, makes potential integration of the ideas from this effort much simpler then it was a few years ago. Thus, a transition with these technologies is plausible. However, UML based modeling has become increasingly popular and may be a preferred approach.

2.3.3 SSAT

The System Security Analysis Tool (SSAT [4], [5]) methodology, created by ORA, helps systems designers who are not necessarily security experts to construct assurance arguments using composability arguments. SSAT enforces and extends an NSA System Security Profiling methodology for constructing assurance arguments. SSAT provides a library of generic security models for systems. A system designer can use these generic models to decide what top-level requirements are necessary to supply a certain type of security assurance on the system, and also to partition those top-level requirements into requirements on the components of the system. Then the designer must ensure that the system to be designed is an instance of the generic model, by discharging various requirements.

The ideas surfaced in this approach may be useful in handling situations in which subsystems may need to be replaced with equivalent, but not identical properties. However, before we deal with this approach more fully, we first need to understand how to control situations in which the parameters are simpler (such as the amount of disk space that can be used for auditing).

This tool was based on CORE and the recent upgrades to CORE should make incorporation of the SSAT ideas and the new ideas of this project tractable. However, as noted above, it might be better to transition the ideas using a UML based approach.

2.3.4 DOORS

DOORS [6] is a requirements traceability tool which provides extensive support for managing shared structured data. Its data model is very general: information is organized

into modules, each of which is a hierarchical collection of objects. Each object has a number of attributes, and structured links to other related objects. DOORS provides a convenient "drag and drop" interface for quickly making links between objects. It also provides a wealth of different options for importing and exporting structured data---as Word documents, Microsoft Project data, HTML, etc. DOORS provides a very flexible and useful framework for collaborative recording and modifying requirements data. DOORS does not assume any particular metamodel. Although this greatly increases the flexibility of using DOORS, it eliminates the possibility of any automated heuristic support in maintaining consistency, or checking for completeness. Such automated support would have to be implemented as third-party "critic" programs that worked from DOORS data. One interesting possibility would be to implement our metamodel in DOORS. This would amount to defining a set of module types, object types, attributes and link types that would be meaningful in our metamodel, and then writing the analysis tools such as dependency checkers as various critics on the DOORS structure. Although DOORS does not provide any type checks or other consistency checks on its structures, it would be possible to write a report generator that created reports on problems found with a solution structure represented in DOORS. The only thing that may be difficult to do using DOORS would be to provide interactive support for making a sensible solution structure. Of course, it can all be done after-the-fact via critics on DOORS export files. Many of our metamodel's concepts are completely missing in DOORS, such as the notion of the distinction between properties and context. But that is to be expected, since it is really a generic tool for managing shared structured data.

2.3.5 RequisitePro/UML

Requirements management tools such as RequisitePro [7] share with EMMA the concern with system requirements and collaborative information management among participants on a project. Requisite Pro supports team-based requirements management. It allows project team members to prioritize, sort, and assign responsibilities for requirements. It also provides dependencies among requirements, which can be used to trace the relationships among parts of a project (or even among different projects) and supports the computation of the impact of a change of requirements. Finally, like EMMA, RequisitePro supports the recording of requirements changes, including who made the change, at what time, and why. While much of the functionality of RequisitePro is similar to that of EMMA, EMMA's notion of a solution structure provides a framework for reasoning about requirements and the components built to satisfy them. Also, EMMA's notion of a sequence of releases provides a temporal dimension for reasoning about requirements that is missing in pure requirements management tools. In these ways, EMMA provides more structure on which to hang requirements and reason about them.

2.3.6 StarTeam

Configuration management tools such as StarTeam [8] provide functionality that is in some ways similar to that of EMMA. Like EMMA, StarTeam is a collaborative information management tool. It keeps track of differences between versions of a software product. It supports future releases indirectly, by tracking the handling of change requests. Like EMMA, StarTeam supports distributed projects. However, the

emphasis of StarTeam is mainly on configuration management, and annotating versions of a software product with information about how it differs from earlier versions (and why the changes were made). In contrast with EMMA, StarTeam is not concerned with planning future evolution of a system, but is instead concerned with documenting past evolution.

3. Example: Field Operations Support Website

3.1 Introduction

This example considers a website that supports combat personnel and strategic decision makers in combat situations. The site provides an umbrella for services that take their inputs from a number of different sources.

The strategic decision-makers referred to, generals and certain field commanders, may little or no knowledge of the low–level requirements of the applications running on the site. Their primary concerns are aspects of the battle.

Administrators supervise the day-to-day running of the website. They are also responsible for understanding the resource requirements and making the actual settings that affect the running of the website. As such they may have little understanding of the strategic requirements of the war.

The example considers the regular operation of the site and then examines some scenarios when the site is attacked and resources become scarce. It tries to show the goal clashes that may occur between the decision makers and the administrators, as they both try to do their respective jobs when conflicting resource requirements arise.

3.2 Regular operation of the site

The website shown in Figure 3-1 supports field operations during combat. It hosts two separate web servers (for security purposes) that host one website each.

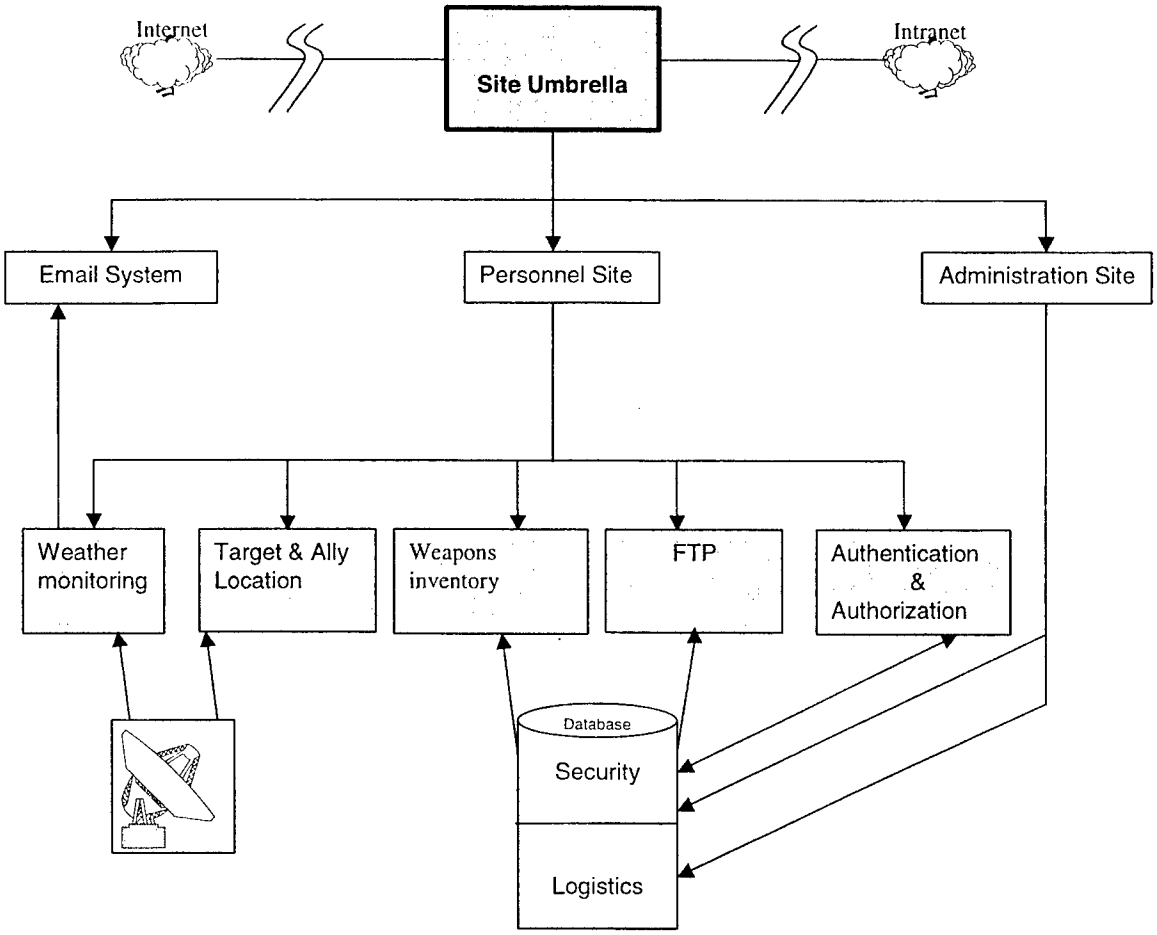

Figure 3-1: Field Support Site Diagram

We briefly describe the components in more detail.

3.2.1 Personnel site

The personnel site hosts the support services for field operations, as shown in Figure 3-1 and are listed below:

- *Weather monitoring System* –Displays weather updates on a map, emphasizing bad weather patterns. It also sends out weather warnings and advisories, by e-mail, to authorized requesting personnel. E-mail updates are independent of the web-site status. The system takes inputs from a satellite and from advisories posted by authorized meteorologists belonging to the administrators group.
- *Target and Ally Location* –Displays the status of troop deployment and enemy targets, indicating the positions of both on a map. It can function in two modes:
  - *Passive* – In the passive mode, the requested map is created as a bitmap and sent down to the user's browser. The user may look at the map and click on it to get zoomed or panned views. This involves making round trips to the server to fetch a new bitmap, which may cause critical delays in the field.

- *Interactive* – In the interactive mode, the client browser launches a program that allows the client to interactively zoom and pan the image at the client end. It also allows field personnel to add their own information to the map and send it back in an encrypted form. This requires that specially encrypted custom HTML headers be sent to the browser to tell it which application to launch. If these headers are compromised, the enemy might be able to figure out which program is being used and simulate the program or insert false headers and create chaos.

- *Weapons inventory* – Displays the current inventory of weapons available at the base and may also be used to file updates on weapons spent in the field. It is linked to the logistics database.
- *FTP Service*– Allows authorized users to write and read encrypted files to the server. Shares a machine with the web server.
- *Authentication and authorization* – Allows personnel to log in to use the site. This service is linked to the security database and can operate only if the security database is up and running.

The web server hosting the personnel site logs to files residing on its own machine.

3.2.2 Administration site

The administration website is hosted on the second web server and provides a set of administration features for managing the web servers, the two websites, and the databases. It is accessible from the intranet and has a very high level of security. It hosts tools for updating the personnel website and the administration site.

3.2.3 Email system

The email system is made up of two outgoing email servers and one incoming mail server. One of the outgoing mail servers sends out email with 128-bit encryption, while the other server uses 64-bit encryption. Given the state of current technology, 64-bit encryption can be hacked, but 128-bit cannot. The email system is used by several parts of the site. In particular, the weather system sends email warnings of inclement weather and personnel use email to communicate.

3.2.4 Users

There are essentially three types of people who will interact with some portion of the site:

- Administrators check for signs of intrusion and pull the switches on the controls. They primarily use the administration site and look at the personnel site only to keep an eye on its working.
- Strategic Decision Makers use several parts of the personnel site and may have access to the administrative site to post updates on the situation or to send out communications.
- Combat Personnel are the primary users of the personnel site. They use it to update their current location on the maps and send them back to the site, as well

as to check on enemy movements, weapons availability, updates on the weather, and any special communications.

3.3 Operating modes of the applications

Looking at the resource requirements of the components and services in various modes of operation will indicate the necessary constraints on the resources for the component to operate in a particular mode.

Component: Web Server

The web server has three modes: Full service to a selected few, Limited service to all, No service.

Mode Name: Full Service to a selected few	
Description: All services up, available to only some selected IPs.	
Resource	**Constraint**
Network connectivity	Set filters and allow only selected IPs in.
Disk Access	20 MB space, enough for the web server to log and to serve out files.
Database Access	Read and write access for both databases.
Mode Name: Limited service to all	
Description: Only a few selected services are made accessible to all. It is assumed that some of the hits to the site will be fake traffic.	
Resource	**Constraint**
Network connectivity	High Bandwidth, no IP filters.
Disk Access	10 MB, enough to serve files and log selected events.
Database Access	Read and write access for both databases.
Mode Name: No service to anyone.	
Description: Only a single page, displaying "Site Not Available" will be served.	
Resource	**Constraint**
Network connectivity	Low bandwidth.
Disk Access	None for internet users.
Database Access	Read and write access for both databases.

Component: Target Location System

The target location system has two modes: Interactive, Static.

Mode Name: Client-side Interactive.	
Description: Client launches interactive application. Browser needs to be sent specially encrypted headers to start the application.	
Resource	**Constraint**
Client side program	Special headers with encryption required.
HTML headers	Header must contain the name of the application to be launched on the client side
Mode Name: Client-side Static.	
Description: In the static mode, the client browser merely interprets the data as HTML and shows the map.	
Resource	**Constraint**
HTML headers	No special headers required.

Component: E-Mail System

The email system has three modes: Fully functional high-encryption, fully functional low-encryption, Incoming only.

Mode Name: Fully Functional with High Encryption.	
Description: Both Incoming and outgoing email allowed, outgoing mail is 128-bit encrypted.	
Resource	**Constraint**
Incoming mail server	Running
Outgoing mail server	High encryption mail server required
Mode Name: Fully Functional with Low Encryption	
Description: Both Incoming and outgoing email allowed, outgoing mail is 64-bit encrypted.	
Resource	**Constraint**
Incoming mail server	Running
Outgoing mail server	Low encryption mail server required
Mode Name: Incoming only	
Description: Only incoming mail allowed. Outgoing blocked.	
Resource	**Constraint**
Incoming mail server	Running
Outgoing mail server	None required

Component: Weather Notification System

The weather notification system has two modes: Fully functional, Update only.

Mode Name: Fully Functional.	
Description: System posts weather conditions on the website and also sends email to notify the soldiers of inclement weather.	
Resource	**Constraint**
Weather Satellite	Unrestricted access
Email system	Must allow outgoing email.
Mode Name: Website update only.	
Description: Email notifications are not sent. The website gets updated every 24 hours.	
Weather Satellite	Unrestricted access
Email system	Not required.

Component: FTP Service

The FTP service has two modes: Fully functional, Write only.

Mode Name: Fully Functional.	
Description: Allows internet users to get and put files.	
Resource	**Constraint**
Disk Space	Space allocation must increase dynamically to allow users to put in new files.
Mode Name: Read only	
Description: Allows internet users to only get files.	
Resource	**Constraint**
Disk Space	Only as much as required by previously stored files is allocated. Allocation remains fixed.
Mode Name: Write only	
Description: Allows internet users to only put files.	
Resource	**Constraint**
Disk Space	Space allocation must increase dynamically to allow users to put in new files.

3.4 Some attack scenarios

To support operation in any particular mode, the resources required by a component or service must satisfy specific constraints. Under attack conditions, resources such as disk space, bandwidth, and database access may be reduced or compromised. Decisions have

to be made about what modes to run the services in, given the quality of service levels available from the resources and the components of the system.

Sometimes, the demands of a strategic decision maker to operate certain components in certain modes may place incompatible constraints on resources. The administrator, in order to resolve the situation, must present the best possible set of working choices to the decision maker, based on the resource constraints. Two such examples are described below.

Loss of an email server

- The high encryption outgoing mail server has been hacked and must be shut down to prevent the attackers from sending out fake messages.
- Therefore, either the low encryption mail server must be used or the mail service must be set to "Incoming Only".
- Because of the possibility of a sudden storm, the General would like the weather system to operate in the "Fully functional" mode.
- The administrator may therefore put the weather system in the "Fully functional mode" and put the email system in the "Fully Functional with Low Encryption" mode or he may use the "Website update only" mode of the weather system and set email to "incoming only".
- The administrator can present the general with two options:
  - Enabled emailed emergency weather updates and risk having messages hacked, which might compromise troop position.
  - Order soldiers to check the website every 24 hours and risk missing a storm warning, if it occurs between checkups.

Denial of service attack

- The FTP service is set to "fully functional" mode and is being deluged by a flood of files being written to the FTP site, as part of a hacker attack. These files are consuming increasing disk space.
- The web server is set to "Full service to all" mode and is running out of disk space to log, as the FTP service is consuming all available disk space, and they share the same storage device.
- Any mode of the web server requires some disk space for logging. If the FTP service is left in the "Fully functional" mode or set to the "write only" mode, the web server would stop operating. Hence the FTP service must be degraded to "Read only" mode.
- The administrator again presents the general with two choices:
  - Risk losing vital field reports, which are sent in via FTP, but keep the website functional, thereby providing field personnel with critical support.
  - Abandon the website in order to keep the field reports.

3.5 Fallback positions

One of the ways to address the conflicts that arise from resource shortfalls is to adopt one of a number of pre-computed fallback positions that produce an acceptable tradeoff between the operational and security requirements.

The following lists some possible fallback positions.

During Denial of service attack:

Settings	Results
1.Activation of filters, denying connections to all but those with special IPs. (e.g. Commanders). Denied personnel can still communicate by Email. OR	1.Target and ally location service banned to a majority. 2.Commanders get full functionality, uninterrupted. 3.Less chances of information being compromised. 4.Communication still possible by means of email access with those cut off.
2.Only filter out IPs suspected of being compromised. OR	1.Greater risk of attack. 2.All services fully accessible to a majority of the personnel.
3.Set no IP filters, but reduce bandwidth support to all but select IPs and Cut off E-Mail Access to all.	1.All services accessible every authorized user. 2.Attack might still cause significant impact to the availability of services to all including the special IPs. 3.Increase in the amount of time to access services for a majority. Longer access times to information for authorized personnel may mean critical delays in the field. 4.Less chances of being spammed.

During Database Break-In:

Settings	Result
1.Offline the logistics database and leave the security database online. OR	1.Chances of security information being stolen exist. 2. Location information, weapons, and inventory unavailable. 3.A hacker may insert false identities into the security database. 4.Legitimate updates to the security database to weed out false Ids and temporarily disable logins of suspected infiltrators are still possible.
2.Change permissions on both DBs to Read-Only OR	1.No updates of information possible. Hence Some real time data may not be available 2.Possibility of security information being stolen 3.Possibility of logistical information being stolen. 4.Website services accessible to legitimate users
3.Leave both databases online.	1.Real time inventory and location updates possible. 2.Possibility of data integrity being compromised by hacker is increased. 3. All services accessible to legitimate users.

A tool could give the administrator the information from the "Results" column, for a combination of settings of the resource switches.

4. Example: Logging Security Events on a User PC

4.1 Introduction

This example considers the relations between two security applications—attack assessment and intrusion detection—and the configuration parameters that control auditing in Microsoft's NT version 4.0.

Intrusion detection systems search the logs for patterns of suspicious activity. Attack assessment tools search for clues that help assessors to reconstruct the actions of an intruder. Intrusion detection is frequently performed after the fact, on historical logs. Attack assessment can require logs that go back even further in time, since sophisticated attacks often take the form of a sequence of very short intrusions spread out in time. Therefore, audit logs should be kept for a fairly long time, to make sure that the extent of an attack can be discovered.

Unfortunately, logging is expensive. Maximal logging can use a high percentage of processor power. Logs maintained over a long period take up a lot of disk space. Therefore, audit logging must compete with other functions.

This example could be made more complex by extending it to monitoring events on hosts that have one or more servers (mail, file, web, etc.), or by increasing the user needs to include the ability to detect attempts to hide attacks (e.g., by erasing the logs) or to detect misuse of administrative privilege. But even this simple case should help clarify the role of the specifications our method requires application vendors to supply.

We consider a networked PC that does not execute any system servers (e.g., file server, mail server). We are concerned primarily with the system security log, which captures a record of events that may affect system security. These include the success or failure of:

- logon/logoff
- file and object access
- use of user rights
- user and group management
- security policy changes
- restart and shutdown
- process tracking[2]

Based on our familiarity with security applications, we believe that logon/logoff, file and object access, and process tracking are the most useful for intrusion detection and attack assessment. We do not require system administrators to know this sort of information. Vendors or domain experts must do this sort of analysis.

4.2 Application modes

To support attack assessment and intrusion detection, logging must log "adequate" information and also maintain the integrity of the logs (e.g., by detecting and debugging log damage). To be adequate, the information must at a minimum allow detection/assessment of the "usual" attacks, including reconnaissance attacks, and must be maintained long enough to detect/assess attacks spread out over time (one year, say.) A reasonable discipline for maintaining the integrity of the logs might include the requirements: back up logs, wrap log files that overflow, do not overwrite information that is not backed up.

A vendor (or domain expert) might therefore describe the logging needs of a fully functional attack assessment or intrusion detection application as follows:

1) Audit major security events

---

[2] Process tracking means logging an event whenever a new process is created. It provides process IDs, which are needed to associate logoff and file accesses with a given user.

a) Audit login/logoff
b) Process tracking
c) Audit access to local files and objects

2) Maintain audit trail for one trailing year
   a) Create log file big enough to hold one month of audit data
   b) Archive log file monthly to file server
3) Back up audit data to guard against log file failure
   a) Back up local log file frequently
   b) Overwrite backed-up log entries if file is full
   c) Do not overwrite entries that have not been backed up

We will specify the resource needs for each mode of operation at this level of granularity. (Section 4.5 considers a refinement of this picture.) For the purposes of this example, we suppose that both intrusion detection and attack assessment require complete audit records. Thus, we don't define degraded modes of operation compatible with shortfalls in (2) or (3)—incomplete records mean the tools won't operate at all.

We define two modes for attack assessment: Fully functional, Login-only. Login-only mode is possible if (1c) is unavailable. It provides a minimal capability: the ability to detect unusual logins by insiders or simply to observe who was logged in when some bad thing happened. Under our assumptions, intrusion detection has only one mode, since knowledge of logins alone would not be useful.

4.3 Configuration parameters

The following configuration parameters are relevant to the needs of attack assessment and intrusion detection: the frequency with which the log file is archived, the frequency with which the log file is backed up, the size of the system log file, the size of the local log file (on the PC), the event filtering (determining which of the seven kinds of auditable events should be recorded), and the policy for log wrapping (discussed below).

Since system logging can consume so much disk space, NT requires the administrator to set a maximum size for the log file. Microsoft provides three choices for event log wrapping when the log file is full:[3]

1) Overwrite events as needed
2) Overwrite events older than N days if log file is full (fill in a value for N), then drop any further new events
3) Do not overwrite events (drop new events)

When the log file is full: choice (1) tells the system to overwrite events starting at the beginning of the file; choice (2) says that the log should wrap up to the point at which events within the last N days would be overwritten—after which, logging will stop; and choice (3) says that, at this point, event logging should stop.

---

[3] None of the choices includes notifying anybody that the log file is full.

4.4 Relating configuration parameters to applications

We next define the relations between configuration parameters and the modes of the applications—by relating the parameters to the requirements for each mode.

In fully functional mode (for either attack assessment or intrusion detection) the filter determining which events to log must capture at least logon/logoff, process tracking, and file and object accesses. In login-only mode the filter must capture at least logon/logoff and process tracking.

The setting of that filter determines how much space will be needed to store N days worth of auditing. Only experience can tell us how to calculate that[4] as function, S(N), of N. The size of the system log file must be at least S(31) and this file must be archived at least once a month. An acceptable size for the local log file depends on how frequently it is backed up. If it is backed up every N days, its size must be at least S(N); and we set the overwrite policy to option (1). (Here, "must" means "for these tools to operate at all.")

Thus we can determine the available modes as a function of the filter setting, the sizes of the logs, and the frequency of backups and archiving.

4.5 A refinement: Reporting

A more refined specification of the applications is necessary if we want to be able to report configuration errors to the cyber commander without also supplying low-level details. The refinement given below distinguishes between the logon/logoff information and the file event information, so that we can explicitly point to the requirements violated if, for example, we stop recording file access events for one hour.

1) Audit major security events
    a)Log logon/logoff information
        i)Record logon/logoff success and failure events
        ii)Record process tracking events
    b)Log file and object access information
        i)Record file and object access success and failure events
        ii)Record process tracking events
2) Maintain audit logs for one trailing year
    a)For each log file, maximum log file size shall be big enough to hold all audit entries during the archive period.
        i)Security log file
        ii)Access information log file
        iii)Process tracking log file
    b)For each log file, archive the log file periodically to a file server

[4] Of course, the amount of audit space required increases as the computer is used more heavily—for example, during a major crisis.

3) Back up audit data in case of log file failure

a)For each log file, back up local log file frequently.
b)For each log file, overwrite backed-up log data with new entries if log file is full.
c)For each log file, never overwrite audit entries that have not been backed up

This set of requirements does not prejudge implementation details such as whether the logon events are kept in the same file as file access events. It also partitions the information so that we can explicitly point to the requirements violated if, for example, we stop recording file access events for one hour.

4.6 Discussion

4.6.1 Better description of applications

Our example describes the applications quite simplistically. Descriptions could be improved both by defining more informative modes of operation and by stating more detailed requirements.

More informative modes. We partitioned modes discretely: the application does or does not do certain things. However, an application occasionally unable to deliver the desired performance characteristics for short periods of time is quite different from one permanently in a degraded state. There might be some way to define the modes so as to deal with lacunae in both the log and resource constraints. An appropriate characterization of application needs might be sufficient to avoid a detailed analysis.

More detailed requirements. We stated an application's requirements as questions with yes/no answers. It would be desirable to introduce the notions of tolerance and approximation into the requirements so that assessments are not unnecessarily pessimistic (forcing into some degraded modes. Note, however, that margins of error may depend on circumstances. For example, keeping an audit log on the file server may normally impose an M% performance penalty on the processor, but if the usual file server has been replaced by a slower substitute or the network is very heavily loaded, the performance hit can be much greater. To handle this well will probably require some system-specific support tool. However, if an application is described with sufficiently fine-grained modes, an approximate solution may be quite satisfactory.

4.6.2 Aggregation

Our descriptions of the applications omitted processing power from the requirements and performance estimates from the definitions of the modes. The actual performance of an application depends on the scheduling of all tasks, whose demands do not add up in any simple linear way. For example, processor scheduling tends to fall apart when total utilization is higher than 70%. Demands for storage are similarly non-linear, though the impact of a high shared load will manifest itself differently. The determination of "disk space needs" is more complex then simply adding up disk space demands. One could measure how a particular application actually interacts with audit logging when they are set at given modes. Doing this in advance, or by simulation, avoids the intractable

problem of figuring out analytically how the performance of the two (application and logging) interacts.

We have considered introducing a notion of an aggregated resource, such as total disk-space. Doing so would be complex. It means one thing if all storage is (transparently) distributed and another if, for example, an audit log is required to reside on the disk of the PC whose actions are being logged. Once again, a system-specific function may be needed to relate application resource needs with the collection of resources available.

4.6.3 User needs

We have given a simple example of assessing system configurations in terms of the application modes they support. Real user needs lie at a higher level. After a cyber attack, is it better to have intrusion detection in normal mode and other functionality in a somewhat degraded mode, or vice-versa? By providing some intermediate representation—the application modes—we hope to simplify this analysis process. Whether to tolerate degraded auditing requires insight into mission needs and the nature of the attack.

5. Example: A Secure Distributed CORBA Application

This example studies reconfiguration of a distributed CORBA application running on a system with degrading resources. We conclude that fallback positions must be developed in advance, because the CORBA Security policies are too complex for a system administrator to modify at runtime. Without the ability to debug and test a new policy configuration, an administrator would find it difficult to guarantee that the whole system would continue to function. We conclude by describing how a range of fallbacks could be developed and presented to the user. Section 5.1 provides an overview of CORBA, section 5.2 provides an overview of the CORBA Security Service and accompanying policies. Section 5.3 describes the distributed application scenario. Section 5.4 describes the kind of support that the developer of the distributed application (not the CORBA ORB or Security Service vendor) could provide to the system administrator.

Sections 5.5 and 5.6 provide a high level description of the system administrator's decision process and then, within the context of the scenario from Section 5.3, describe potential fallback positions. In Section 5.7, we discuss what kinds of help could be automated and point out places where automation would be difficult.

5.1 CORBA overview

The Common Object Request Broker Architecture (CORBA), introduced in 1992 by the Object Management Group (OMG), has gained widespread acceptance as a distributed object computing infrastructure. CORBA is a conceptual "software bus" that supports an environment of plug-and-play software components. Applications communicate with each other regardless of the platform they are running on, the language in which they are written, and their location. The Object Request Broker (ORB) is the fundamental CORBA mechanism used to provide the communication channels.

5.2 CORBA Security Service overview

The CORBA Security (CORBASec) specification addresses the security requirements of the CORBA distributed object environment [10]. In general, CORBASec describes the security association between a CORBA client and a CORBA server. It addresses all areas of security, including identification and authentication, authorization and access control, security of communication, security auditing, and the administration of security information. Figure 5-1 below depicts the basic high-level CORBA security model.

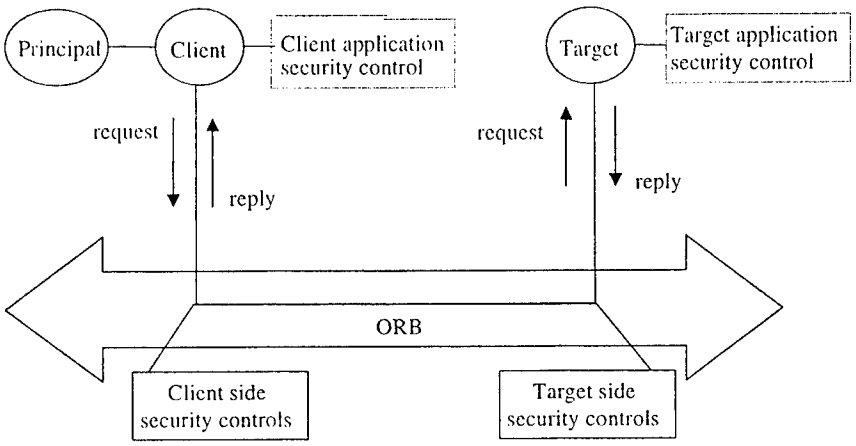

Figure 5-1: Basic CORBASec model

Each object resides in one or more Security Policy Domains. Every domain has an Invocation Access Policy that defines the privileges each principal has when attempting to access objects in that domain. These privileges are called the *effective rights* of the principal. A principal may have different effective rights in different Security Policy Domains.

Every interface associates *required rights* with each of its methods, specified either as "All S" or "Any S," where S is a list of rights. "All S" means that access is allowed only to those principals whose effective rights include every right listed in S. "Any S" means that access is allowed only to those principals whose effective rights include at least one of the rights listed in S. These access rules apply to every object that is an instance of the interface regardless of the domain in which the object resides. CORBASec has defined a small set of standard rights, called the CORBA family of rights. These are {g (get), s (set), m (manage), and u (use)}.

When a Client invokes a Target, the ORB determines whether to permit the invocation, thus providing control over accesses to "security-unaware" objects without requiring any changes to the interfaces or implementations of those objects. (An object is security-unaware if it provides no security functionality of its own.) These access decisions, made

on the Target side of the invocation, are based on three things: received *credentials*, required rights, and effective rights.

On the Client side, an authentication mechanism (e.g., SPKM, Kerberos, CSI ECMA, SESAME) establishes the identity of a principal (human or entity). These are incorporated into the credentials passed to the Target side ORB.

The Target side makes the access decision straightforwardly: by consulting the Target domain's Invocation Access Policy, the ORB determines the effective rights of the principal whose credentials it has received. Access is granted if and only if the effective rights of the principal satisfy the required rights. It should be noted that membership in a particular Security Policy Domain also provides automatic enforcement of other policies including: Audit Policies that control which operations on which objects are to be audited; Delegation Policies that specify which delegation model will be allowed for delegation of privileges in a multi-tier client/server environment; and Invocation Policies that specify quality of protection (encryption mechanisms, etc.) requirements.

CORBASec 1.2 specifies standard features for a security service. These features are structured into several packages. The main security functionality is provided by two of them:

- **Level 1**: provides a first level of security for applications that are security-unaware and for applications with limited needs to enforce their own security by access controls and auditing.
- **Level 2**: provides more security facilities, and allows applications to control the security provided at object invocation. It also provides facilities for applications to administer security policy.

To use either Level 1 or Level 2 security packages, the system administrator creates the users and groups, sets required rights on CORBA interfaces and methods, and creates security domains and associated policies.

5.2.1 Policy types

A security domain supports five types of policies:

- *Invocation Access Policy.* This policy specifies the effective rights for users or groups with respect to objects in a particular domain.
- *Delegation Policy.* This policy specifies the kind of delegation used by objects in the security domain. There are three types of delegation: none, simple, and composite. No delegation means that the target receives only the credentials of the immediate caller. Simple delegation means that the target receives only the credentials of the original caller. Composite delegation means that the target receives the credentials of all parties in the call chain.
- *Client Secure Invocation Policy.* This policy specifies which security features are supported or required for outgoing calls on the specified interfaces. These security features include confidentiality and the type of delegation (none, simple, composite).

- *Target Secure Invocation Policy.* This policy specifies which security features are supported or required for incoming calls on the specified interfaces. Security features include confidentiality, and type of delegation (none, simple, composite).
- *Audit Policy.* This policy specifies events to be audited at a specified interface.

5.2.2 Access control example

We illustrate how access control works by a simple example. Consider the following CORBA interfaces for a bank:

```
Interface Bank {
   attribute string name;
   attribute AccountList accounts;

   account createAccount (in string name, in string password);
   float totalAssets();
}

Interface Account {
   attribute string name;

   void changePassword(in string oldPassword, in string newPassword);
   float deposit(in float amount);
   float withdraw (in float amount);
   float balance();
}
```

We set the required rights for these methods as follows:

Interface/Method	Required Right
Bank::createAccount	Any corba:g
Bank::totalAssets	Any corba:m
Account::changePassword	Any corba:m
Account::deposit	Any corba:m
Account::withdraw	Any corba:m
Account::balance	Any corba:g

There are four users:

- BankExecutive
- Teller
- Jane
- Bob

Figure 5-2 shows three security domains and the operations that each user should be able to perform on the objects. The BankExecutive should be able to get information about the bank's total assets, but he should not be able to create an account. The teller should be able to create accounts at the bank and perform operations on the accounts. Jane and Bob must go through the teller to deposit and withdraw funds or to change their password. Users may get information about their own account balance.

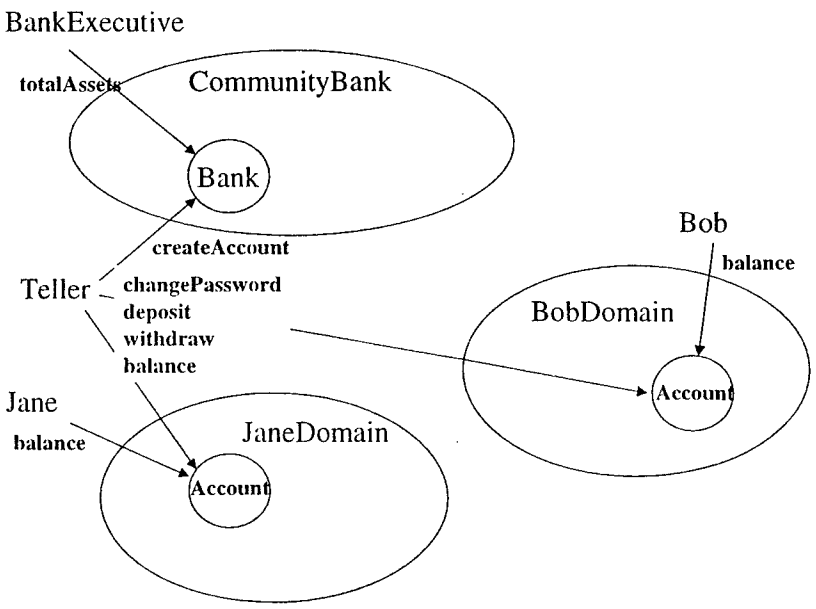

Figure 5-2: Security domains and objects for the bank example

Here are the Invocation Access Policies specifying the rights for each user in each Security Domain:

CommunityBank	BankExecutive: corba:m
	Teller: corba:g
JaneDomain	Teller: corba:m
	Jane: corba:g
BobDomain	Teller: corba:m
	Bob: corba:g

The Teller has the corba:g right in the CommunityBank domain, so he has sufficient rights to perform the createAccount method on the Bank object. The Teller has insufficient rights to get information about the total assets of the bank. The BankExecutive has the corba:m right in the CommunityBank domain, so he does have sufficient rights to get information about total bank assets. The BankExecutive does not have sufficient rights to create an account.

Jane has the corba:g right in the JaneDomain, so she has sufficient rights to get access to her account balance. She does not have any rights in the BobDomain, so she is not allowed to get information about Bob's account.

5.3 Secure distributed CORBA application

Consider a complex CORBA-based distributed application having multiple servers on multiple hosts. Legacy applications, including databases, are integrated by using CORBA wrappers. Client applications are available from a number of geographically dispersed locations. See Figure 5-3 for an overview of the system. Each box in the diagram represents a different host; and all hosts are connected to a network.

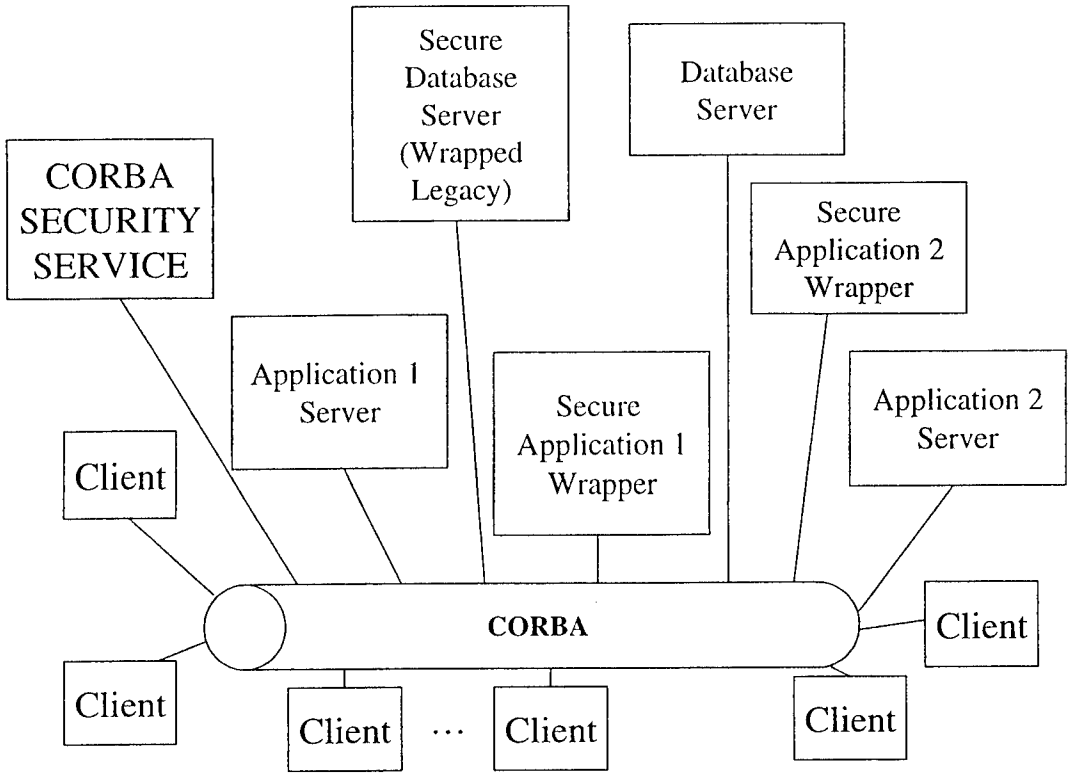

Figure 5-3: Example secure distributed CORBA-based system

One CORBA Security Service provides authentication, access control, message protection and auditing for all systems. The "security configuration parameters" for using the CORBA Security Service are the definitions of groups, domains, policies, and required rights.

In this example, we assume that the CORBA Security Service parameters, in the form of policies, can be set to guarantee appropriate access and message protection. Figure 5-4 illustrates that during normal operations, clients must go through the Secure Application Wrappers and may not call the legacy Application Servers directly.

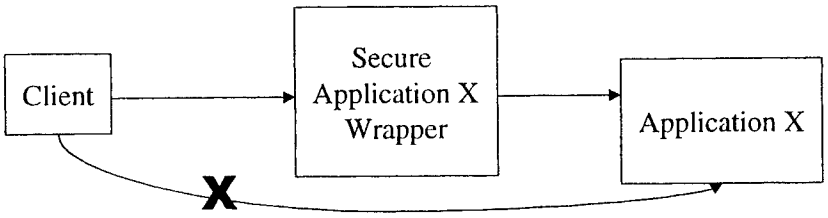

Figure 5-4: Under the auspices of the CORBA Security Service, a client cannot call an application server directly

Several system administrators cooperate to provide system support. If the system were being attacked, the system administrators would have to consider a multitude of fallback positions and associated tradeoffs.

5.4 Support for the system administrator

The system administrator has ways to monitor the system's performance. During or after a cyber attack, the system administrator needs to be able to cope with changing system characteristics. We suggest that the developer of the application could offer some assistance to the system administrator.

The developer could prepare a set of operating modes for the distributed application. For the secure distributed CORBA example, we can imagine four modes that would support different levels of security, such as shown in Figure 5-5, and allow the system administrator to select the appropriate level of security.

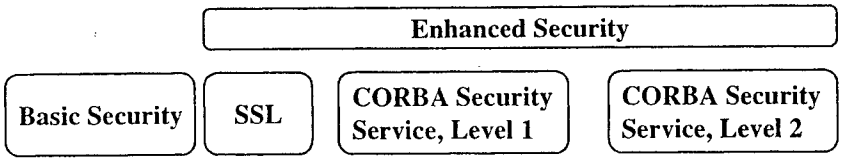

Figure 5-5: Possible levels of security for the distributed application

The levels are:

Basic Security: Applications run on a plain CORBA ORB. Neither the CORBA Security Service nor the CORBA SSL enabled ORB are running in this mode. Users and groups may be given launch and invoke privileges for server and client applications.

SSL: The Secure Sockets Layer ORB (CORBA SSL) is enabled, providing encryption. CORBA SSL is a much lighter weight service than the CORBA Security Service. We will not run the CORBA Security Service and the CORBA SSL ORB at the same time.

CORBA Security Service:

Level 1: Provides confidentiality, integrity, and accountability for security unaware applications. Security unaware applications can also run in insecure mode on a plain CORBA ORB.

Level 2: Provides confidentiality, integrity, and accountability for security aware applications. "Security aware" application code handles some of its own security and depends upon the CORBA Security Service. For a security aware application to be able to function at any of the other levels, there would have to be the equivalent of a system environment variable that would act as a flag to the application to turn off the security related code.

The Basic Security setting, requiring only a plain ORB, requires a modicum of system resources. At the other extreme, the CORBA Security Service is very resource intensive. Because so many objects are needed to support the various security policies, a lot of processing activity goes on behind the scenes. Some of this processing depends on how the policies are set up, but even in simple cases, running the CORBA Security Service imposes a noticeable performance penalty.

In addition to developing different levels of security support, an application developer or security administrator can develop multiple fallback positions for each mode. We provide examples of fallback positions in Section 5.6.

With support from the application developer, the system administrator has two important pieces of information: the level of processing that the system can support, and options for running the application in different modes/fallback positions. The system administrator needs a way to map what the system can support to a particular fallback position. It is this mapping that a requirements management tool could provide.

5.5 The System Administrator's Decision Process

During a cyber attack, the system administrator should be prepared to handle different levels of system degradation. Figure 5-6 starts with an undamaged system at the bottom. With increasingly severe attacks, system resources gradually become unavailable so that, at the top of the illustration, it may not be possible to guarantee a high degree of security, especially at a fine granularity.

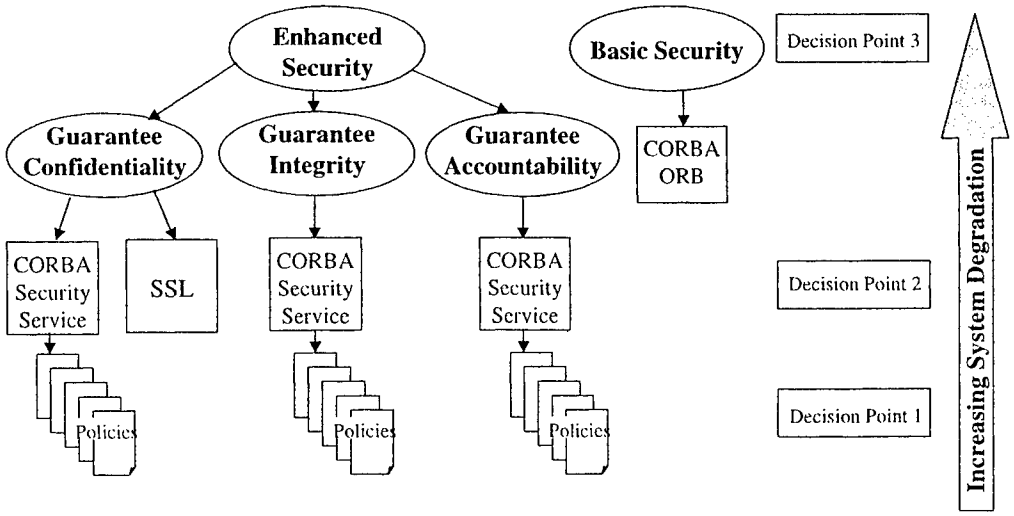

Figure 5-6: System administrator decision points

Decision Point 1. If a stable system is neither under attack nor failing, all of the system resources are available to fully support the distributed application. The system administrator has the option of providing security via the CORBA Security Service. The result of Decision Point 1 is to determine the initial configure the CORBA Security Service. The system administrator must choose a cohesive set of policies. It is not possible, for example, to configure the security service with one set of policies to guarantee integrity and a different set of policies to guarantee confidentiality. For a high-level description of the CORBA Security Service and the different types of policies, see Section 5.2. The main difficulty involved in developing a set of policies is that the policies are interdependent, so that a change in one policy often requires a change in others. It is very easy to define mistaken policies, causing the CORBA Security Service to deny applications access to functions they require.

This complexity makes it risky to change settings on the fly. We envision that an application vendor or security administrator could develop sets of policies that would correspond to different fallback positions. The requirements management tool would describe the pros and cons of the various fallback positions so that the system administrator would be able to make a good decision.

Decision Point 2. Imagine that the system is under attack and system resources are degrading. The system administrator has the option of using the CORBA Security Service, but perhaps with coarser grained policies, or using CORBA SSL, which only provides confidentiality. CORBA SSL is a much lighter weight solution than the CORBA Security Service. A damaged system might not have enough resources to run with the full overhead imposed by a CORBA Security Service. A useful decision aid would help the system administrator realize the impact of choosing CORBA SSL.

Decision Point 3. If the system is severely damaged, the system administrator must decide whether to run the application with minimal security guarantees. A useful decision aid would help the administrator understand the tradeoffs. Perhaps some

functionality is critical and a limited subset of applications could be run with minimal security. A vendor or security administrator could develop a fallback position to identify those servers and clients that could be launched in a less secure mode.

5.6 Example fallback positions

We elaborate the rather generic distributed application described in Section 5.2, and define various fallback positions for each of the security modes. We first describe the organization of users and groups. This is controlled by needs of the various groups, and not something that a system administrator can do much about when responding to a degrading system. Because of the number of users involved and the difficulties associated with key distribution, configuration of users and groups should be a fairly stable aspect of the CORBA Security Service configuration.

In our example scenario, there are 2000 users, "user1" – "user2000". We associate the users with the groups illustrated in Figure 5-7:

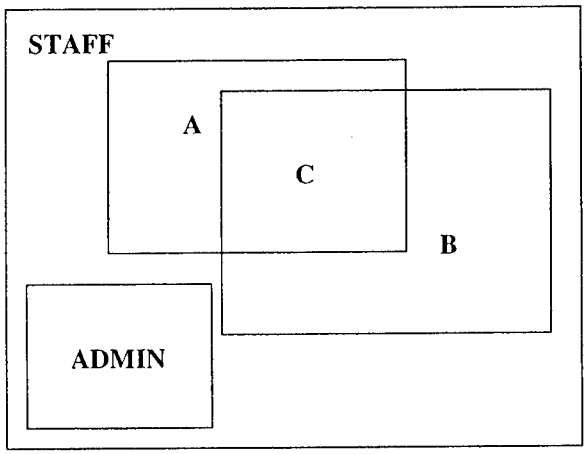

Figure 5-7: Groups of users of the sample distributed application

Users are assigned to the groups as follows:

STAFF: "user1" – "user2000"
A: "user1000" – "user2000"
B: "user500" – "user1500"
C: "user1000" – "user1500"
ADMINISTRATOR: "user1" – "user50"

Users assigned to group C are also assigned to group A and group B. Some users in the STAFF group belong to no other group. The groups A, B, and C represent teams of people working on different projects.

In addition, we create a special user, "SERVER_ADMIN". This is not a human user, but an entity that owns server processes. The SERVER_ADMIN user is not assigned to a group. We can give the SERVER_ADMIN user different privileges from those of any

group. In reality, a system administrator would run the processes as the SERVER_ADMIN user. However, if we put the SERVER_ADMIN user into the ADMIN group, the processes could run with too many privileges.

5.6.1 CORBA Security Service Level 2

An environment variable, LEVEL_TWO, indicates whether the system is running in mode CORBA Security Service Level 2. This mode is meaningful only if both clients and servers are security aware. The client code calls the CORBA Security Service to conduct authentication. Server code calls the CORBA Security Service to implement composite delegation. At runtime, client and servers check LEVEL_TWO to determine whether their security code should be executed.

We describe two security policy configurations. The first is the ideal configuration for the fully functioning system. The second represents a fallback position that uses less system resources by supporting fewer security domains.

Ideal configuration. In order to get fine granularity of access control, the ideal configuration includes multiple security domains. We can support separation of activity and different levels of access based on "need to know."

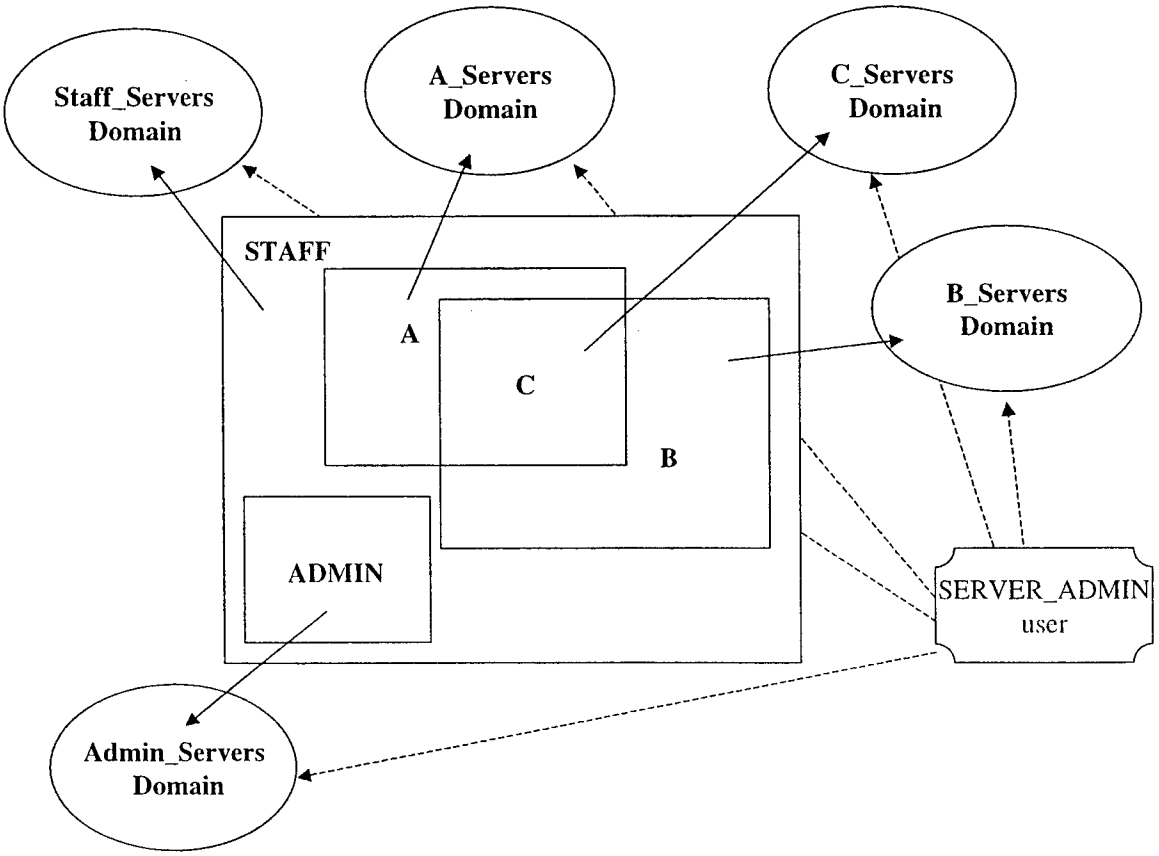

Figure 5-8: Ideal mapping of groups and users to security domains

We define six domains: Five of them (called Staff_Servers, Admin_Servers, A_Servers, etc.) correspond to the groups of users we have defined. Servers run in these domains. The sixth, not shown in the figure, is called Client_Apps, and supports policies for running client applications. We want all members of the Staff to have access to the servers in Staff_Servers; all and only members of group A have access to the servers in domain A_Servers; etc. We could accomplish this formally by setting the Invocation Access Policy of A_Servers (for example) to assign effective right "g" to all and only the member of group A and by setting the required rights for each method on a server to be "All g."

In the ideal configuration, composite delegation could work as follows. Suppose we do not want a client of an A user to call some application X directly. The A Client must first call the Secure Application X Wrapper that is running in the Staff_Servers domain. We specify that the A Client and the Secure Application X Wrapper allow composite delegation. Then Application X code checks the credentials list to make sure that the SERVER_ADMIN credentials are present before completing any request.

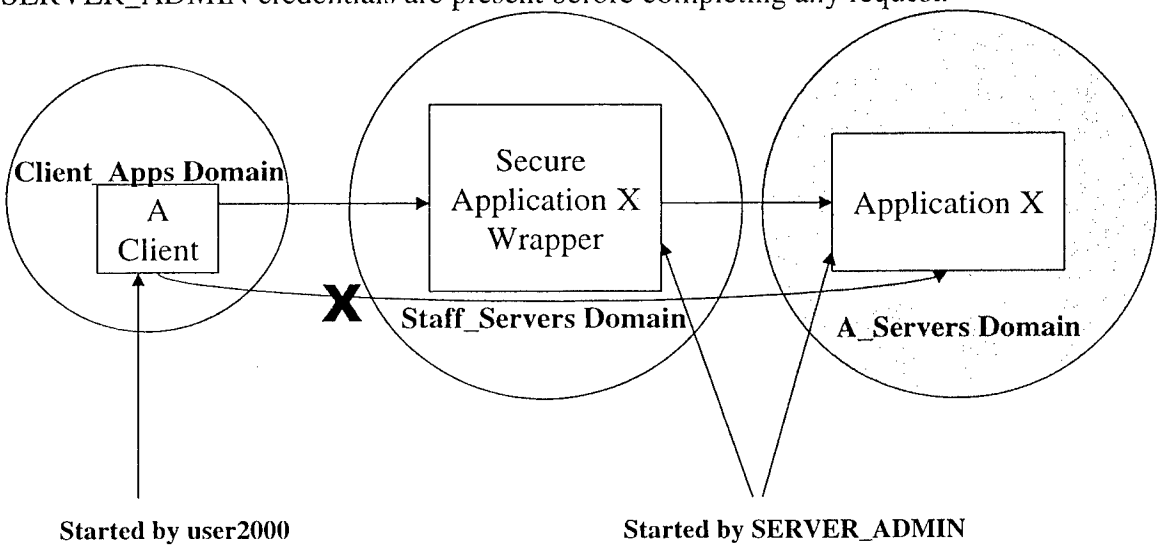

Figure 5-9: Composite delegation

Fallback position. The fallback position in the Security Service Level 2 mode is to support fewer security domains, as pictured in Figure 5-10: Staff_Servers, Admin_Servers, ABC_Servers, and Client_apps. The result is less fine-grained access control. There is no difference between the access permissions of users in the A, B, and C groups. Users belonging only to the STAFF group have no access to applications running in the ABC_Servers security domain.

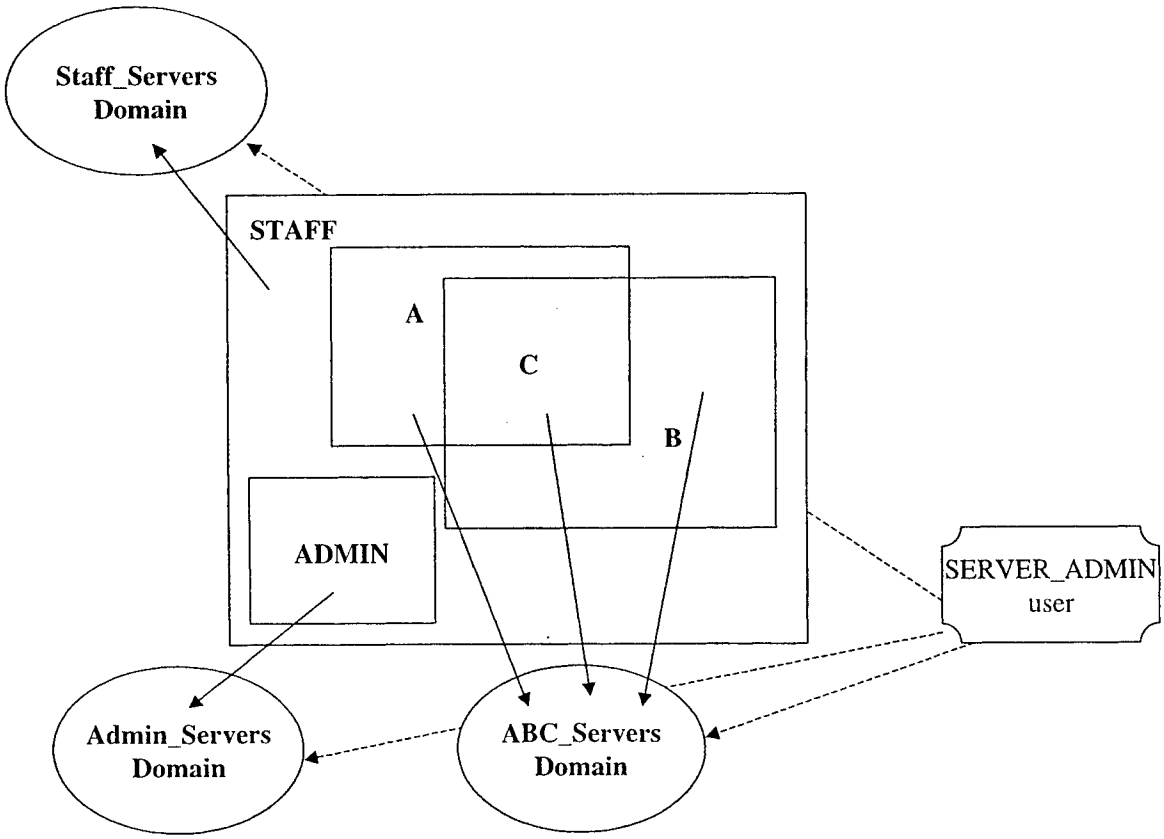

Figure 5-10: Fallback position, supporting fewer security domains

5.6.2 CORBA Security Service Level 1

The LEVEL_TWO environment variable is not set in the CORBA Security Service Level 1 mode. Clients and servers behave as if they are security unaware—they make no calls on the CORBA Security Service. Since composite delegation depends on the ability to run security aware code, composite delegation is not possible in Level 1.

Without composite delegation, we are forced to use a different access control model. Figure 5-11 shows that we can achieve some separation of access to an application X by running three application X Servers, each in a different security domain. But the way in which these Application X Server are launched differs from the way they were launched in the previous Level 2 example. Imagine that an Application X Server has access to a database for its persistent data store. The information in the database is partitioned so that the Application X Server in the A_Servers security domain may not access the same information as the Application X Server in the B_Servers security domain. This shows that moving from one fallback position to another may not be a minor step. The details must be worked out in advance because we must change not only the security domain policies, but also the way that the applications are launched.

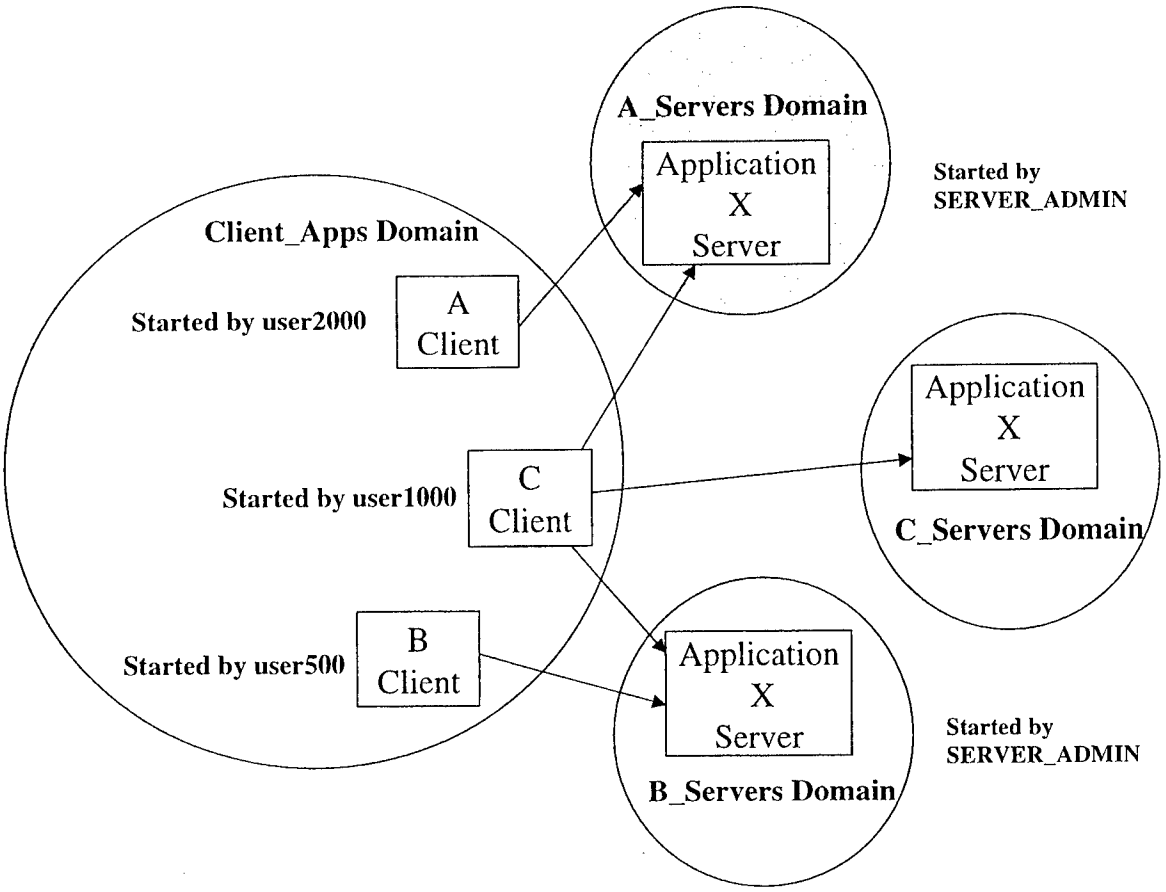

Figure 5-11: Level 1 access model

We can define two fallback positions, analogous to the two fallbacks for Level 1, by defining different numbers of security domains. Switching to one of these fallbacks from Level 2 would require the system administrator to kill some of the processes and launch new processes.

5.6.3 CORBA SSL

If a loss of resources makes it impossible to use the CORBA Security Service, the remaining resources might still be able to support CORBA SSL. CORBA SSL uses encryption for point-to-point message protection; it does not support fine granularity of access control.

Some of the applications designed for the ideal configuration cannot be run in CORBA SSL mode. Figure 5-12 illustrates the constellation of servers and clients that run with CORBA SSL.

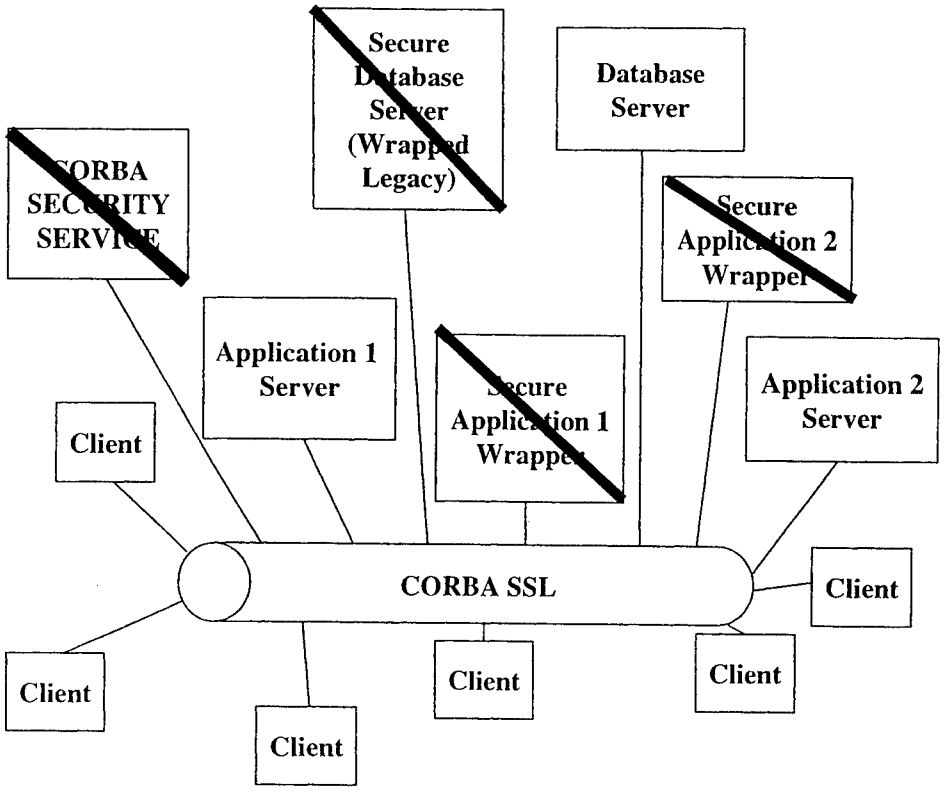

Figure 5-12: Constellation of applications in the CORBA SSL mode.

CORBA SSL runs together with the plain CORBA ORB and offers some facility for access control. The system administrator can specify privileges for users and groups to launch CORBA objects and and/or invoke CORBA methods.

5.6.4 Basic security

System resources may be so degraded that it is not possible to run either the CORBA Security Service or CORBA SSL. This means that no integrity protection (encryption) or fine-grained access control is available.

The fallback position, shown in Figure 5-13, is to run the plain CORBA ORB with only a limited subset of servers and clients. The gray area in the diagram depicts a group of users in a physically secure location. We set invoke and launch privileges to prevent some groups of users from running clients.

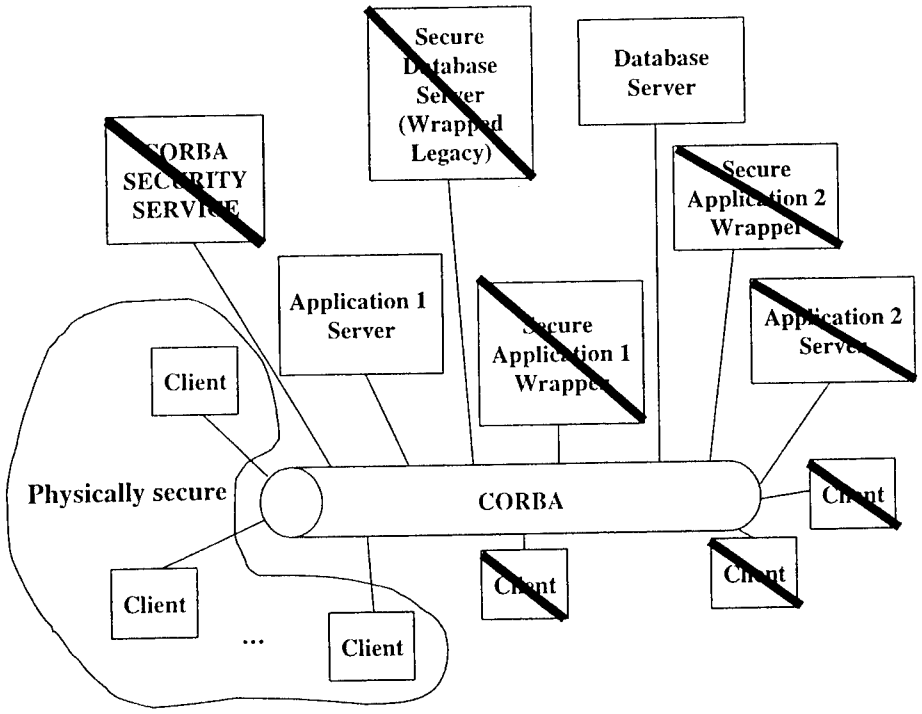

Figure 5-13: Servers and clients that can be launched in the Basic Security Mode

5.7 Automation

A requirements management tool could help by providing:

- a way to match a set of system resources to the demands of a fallback position
- ways to make tradeoffs between various fallback positions.

Which aspects of the problem could be automated? We assume that it is relatively straightforward for a system administrator to assess system capability—to figure out the current demands on the system and the number and kinds of additional processes that the system can support. Automated support for assessment is available: network and system monitoring tools are good examples.

It is not easy for the developer of a secure distributed CORBA application to define fallback positions that are "generic." A sensible configuration will depend on the specific needs of users and groups, the level of protection the information requires, and the number of options the application developer has built into the application. An application developer must work with a security administrator to figure out which fallback positions are acceptable. The fallback positions and tradeoffs must be determined by hand—this part of the process is not automatic.

Once the system administrator has chosen a fallback position, there remains the problem of how to make the change of mode. Some mechanism would have to be in place to shutdown current processes gracefully and start them up in a new mode. This would

require some central console for the network and some way to control processes on remote hosts. Communication between hosts would have to be secured so that new vulnerabilities were not introduced. This procedure could be automated, but any change of mode could present problems for 24/7 services.

6. Formalism

Our goal in this section is to present a preliminary formalism for describing the relations between resource constraints and capabilities, with the goal ultimately of integrating these descriptions within general modeling frameworks such as UML or CORE.

Section 6.1 introduces some basic vocabulary; Section 6.2 can be regarded both as an introduction to the grammar and as a brief discussion of how to build a model from this vocabulary; and Section 6.3 contains a YACC grammar for writing system descriptions in that vocabulary. Finally in Section 6.4 we briefly describe potential analysis support.

6.1 Vocabulary

For any system **S** we define *sets of requirements*, phrased in terms of *assets*. Assets encompass capabilities, features, behaviors, etc. Some assets are discrete (e.g., I/O ports); others are continuous (e.g., bandwidth); still others are discrete but better modeled as continuous (e.g., memory). A principal concern of this section is allocation of assets among applications.

Sets of requirements are grouped into a *family*: at any given time, only one set in the family is *active*, though the active set may change. For example, in response to an external attack, we may degrade our active requirement set to some achievable fallback position.

The assets of **S** depend on how it is configured—that is, on the values of certain *configuration parameters*. We use the assets to satisfy the active requirement set for **S**—either directly or indirectly, by supporting *applications* that provide additional assets by running on **S**. The assets available from the configuration **S** are *configured assets*, and those provided by applications running on **S** are *generated assets*. Assets provided by the external environment, which exist independently of **S**, are *external assets*.

Each application runs in one or more *modes*, and both consumes and provides assets. The *input requirement set* of an application running in a given mode is a set of statements describing the assets required for running in that mode, together the with relationships among the system, generated, and external assets required for the accomplishment of the application. The *output requirement set* of an application in a given mode is a set of statements describing the assets that it provides, together with the relationships among assets that it brings about.

The complete collection of applications running or proposed to run on the system **S** is a *troupe*. Not every troupe can actually run on **S**: there may be insufficient assets; some of the applications may be incompatible with others. A *scenario* is a snapshot of

applications in progress, their modes, the assets allocated to them, and the underlying configuration of the system. (Note: this is not the same as a UML scenario.)

6.2 Modeling

The next section presents a YACC grammar for system descriptions expressed in the vocabulary of section 6.1. Here is the motivation for, and an overview of, that grammar.

6.2.1 Systems

A `SystemDeclaration` states the name of the system, its class (which defines all possible configurations and states of the system), and the system's current instance system (representing the current condition of the system). The system designers and integrators define the `SystemClass`. A system administrator will describe the `SystemInstance`. Interplay between the possible conditions and the actual condition will support inference of ways to improve the condition of the system.

The system condition has four major constituents: configuration, assets, requirements, and applications. All of them are subject to modification in responding to a cyber attack.

6.2.2 Configuration

Configuration consists of in organizing a system, both physically and logically. Connecting processors by cables, installing a firewall host in a selected position, issuing commands during boot-up so as to mount file systems, and choosing security policies to enforce are all part of configuration. No matter what manner of configuration is being represented, the common representation is by the value of a *configuration parameter*. Configuration parameter values may be numeric, both integer- and real-valued, or enumerated identifiers. For example,

```
EventLogging = On
LogCapacity = 5 Gigabytes
```

The full range of values for configuration parameters appears in the SystemClass section. The particular values in force appear in the SystemInstance section.

Security parameters are configuration parameters. The policies that could be in effect and the particular policy that is in effect are all given by means of configuration parameters and their values. Because security is special, we make a distinction between `Functional` and `Security` in the `ParameterKind` child of a `ParameterDeclaration` production.

The values of configuration parameters determine and constrain the assets available to the system.

6.2.3 Assets

An `AssetDeclaration` describes a useful entity that may be generated and/or consumed with the system. Assets are quantifiable, originate in several ways, are of

several kinds, and have various associated rights, constraints, and requirements. Assets exist at one of two levels of abstraction: administrative-level and user-level.

6.2.3.1 Asset quantities

Each asset is a finite entity. There is always a number expressing how much of the asset is available. For example, 10 Megabytes of RAM is an asset, as is 3.5 months mean time to failure for the system. A capability such as the ability to read email is also an asset: it is assigned the quantity 1 if it is available; 0if it is not.

6.2.3.2 Asset origins

Assets can originate externally, may be configured with the system, or may be generated by an application running on the system. External assets are such things as electrical power, network connections to the outside world, a time-of-day service, etc. Configurable assets are such things as disk storage, Ethernet bandwidth, firewalls, etc. Generated assets are such things as databases (which require running applications to be useful), mail-reading capability, and software fault-tolerance.

We support three asset kinds: resources, capabilities, and features. A resource is a component of the system that holds part of the system state.

6.2.3.3 Asset rights

Applications have the rights to use assets different ways. Shared assets are freely usable. Multiple uses of shared assets do not interfere with one another.

Exclusive assets are usable by one application at a time. An application using an exclusive asset has access to all of the asset.

Divisible assets can be apportioned among the players, each having exclusive rights for the portion. Once the full quantity of a divisible asset has been apportioned to applications, no other applications may use the asset until some portion of it is freed.

6.2.3.4 Asset constraints

The quantity available for an asset may be a given, as in the case of external assets, but it may be under the system administrator's control, as in the case of configured and generated assets.

The values of configured assets depend on combination of settings of configuration parameters. Each combination is sufficient to bring about the stated value; it is necessary for at least one combination to hold for that value to obtain. The `ParameterSettingsDisjuncts` production unfolds to describe all the combinations. The `ParameterSetting` production describes a particular combination.

The values of generated assets are described with the applications that generate them, in the `OUTPUT` requirements of the applications.

6.2.3.5 Assets and requirements

Assets figure fundamentally in the requirements placed on the system and on its applications. These requirements are stated in terms of the values of assets, their availabilities and interrelationships. See below under **Requirements** and **Applications** for more information.

6.2.3.6 Asset abstraction levels

At the administrative level, a system is viewed very concretely. Its components are such things as cables, processors, firewalls, parameter settings, and—providing the bridge to the user level—applications. The user-level view of the system encompasses such things as databases, word processor documents, mail readers, and of course, the applications providing them that rest upon the administrative world. The requirements placed on applications state the relationships between assets at the two levels of abstraction.

6.2.4 Requirements

`RequirementSets` are basic to the system. Each member `RequirementSet` is a specification of input and output conditions that could govern the system. The point of our requirements management activity is to find, after a cyber attack, the "best" set of requirements that it could satisfy, possibly after reconfiguration, reallocation of assets, and change of applications.

Some requirement sets apply to the entire system, and are attached through the `RequirementSets` child to the `SystemClass` production. Other requirements apply only to applications, and are attached through the `UserRequirements` and `AdminRequirements` children to the `Application` and `Mode` productions.

The requirements applying to the entire system concern the decision maker. The application requirements are used by the administrator to fit together scenarios that will satisfy or at least inform the decision maker.

Requirements are phrased in terms of the values of assets. Demands are made for certain combinations of quantities of assets (required asset conditions); assurances are made that certain other combinations of quantities of other assets will be available (provided asset conditions). Relationships among values are asserted, both in required and provided conditions. If there is more than one way to bring about a desired effect, disjoined requirement sets are used.

6.2.5 Applications

Applications represent programs running on the system. They typically both require and generate assets. Their administrative-level requirements are specified separately from their user-level requirements; however, the user-level requirements may mention the administrative-level assets, if desired, so as to bridge the abstraction levels.

The requirements attached to the `Application` production are common to and distributed over all the modes of the production (except `NULL`, described below). The

requirements attached to a Mode production are conjoined with the common requirements for the application when it is running in the mode in question.

The special mode NULL represents a non-running activity. It has no requirements, neither consuming nor generating assets. The activity's common requirements are not distributed to the NULL mode.

Abstractly, applications could also represent human activities concerning the system, or hybrid human/machine interactions.

6.2.6 Integration

System integration is made more difficult by the fact that allocation of assets is often nonlinear. For example, the additional processing demands that result from launching a new application will depend on what applications are already running—since the new application not only consumes resources for its own needs, but also adds overhead costs. In practice, it seems likely that this sort of nonlinear aggregation can be represented only by rough, and conservative, approximations. Our grammar contains a placeholder that represents the following model of aggregation: divide the space of configurations into regions (a relevant parameter might be the number of applications running concurrently or the total utilization of the processor) and approximate aggregation on that region by some empirically determined linear function.

6.3 A grammar

```
/* Tokens whose names are in all upper case are for keywords.  */

%token
   A
   ADMIN
   AFFECTS
   AND
   APPLICATION
   APPLICATIONS
   APPROXIMATION

   ASSET
   ASSETS
   CAPABILITY
   CONFIGURATION
   CONFIGURED
   CONTAIN
   CONTAINS
   DIVISIBLE
   DOES
   ENVIRONMENT
   EXCLUSIVE
   EXTERNAL
   FEATURE
   FUNCTIONAL
   GENERAL
   GENERATED
   HAS
```

```
    IN
    INPUT
    INSTANCE
    INTEGRATION
    INTERCEPT

    INTERSECT
    INTERSECTS
    IS
    MODE
    NOT
    NULL
    OF
    OR
    OUTPUT
    PARAMETER
    PROVIDES
    REQUIRE
    REQUIREMENTS
    REQUIRES
    RESOURCE
    RUNNING
    SECURITY
    SHARED
    SLOPE

    SUBSET
    SUPERSET
    SYSTEM
    USER
    VALUE
    VALUES
    WHEN

/* Tokens with mixed capitalization are for standard lexical elements
 */

%token EnumeratedValue
%token Identifier
%token Number

%%

/* The root production comes first */
SystemDeclaration:
    ;

/* Some preliminary vocabulary
 */

DiscreteValue:
    Number
    | EnumeratedValue
    ;

EnumeratedValues:
    EnumeratedValue
```

```
    | EnumeratedValues "," EnumeratedValue
    ;

NumberRange: "[" Number "," Number "]" ;

DiscreteValues:
    EnumeratedValues
    | NumberRange
    ;


/*************************************************************************
                Systems
 *************************************************************************/

/* A SystemDeclaration gives both the full potential of the system,
 * via the SystemClass child, and the current condition (the
 * combination of configuration and state) of the system, via the
 * SystemInstance child.  Any particular condition of the system is
 * derivable from the SystemClass.  The SystemInstance encapsulates
 * the actual condition for convenience.
 * */
SystemDeclaration:
    SYSTEM          Identifier
    GENERAL         SystemClass
    INSTANCE SystemInstance
    ;

/* The SystemClass gives the full potential of the system.  Any
 * particular condition of the system can be derived from it.
 *
 * First, describe the assets available from the external environment
 * (e.g., power supplies, optical cables).
 *
 * Next, give the configuration parameters for the system (these
 * include the security parameters).  Their possible values are
 * affected by the configuration parameters.
 *
 * Then describe the assets that are made available and/or constrained
 * by the configuration parameters.
 *
 * Then give the family of possible requirement sets for the system.
 * The motivation for having more than one set is to have fallbacks in
 * case of system degradation.
 *
 * Then describe the possible applications for the system that may be
 * run to fulfill the requirements.
 * */
SystemClass:
    ENVIRONMENT     EnvironmentDeclarations
    CONFIGURATION   ConfigurationDeclarations
    ASSETS          AssetDeclarations
    REQUIREMENTS    RequirementSets
    APPLICATIONS    ApplicationDeclarations
    INTEGRATION     IntegrationRequirements
    ;
```

```
/*********************************************************************
          External Environment
 *********************************************************************/

/* Certain assets are available independently of the system.  Declare
 * them here.  For more about assets, see below.
 * */

EnvironmentDeclarations:
   | EnvironmentDeclarations EnvironmentDeclaration
   ;

EnvironmentDeclaration:
   EXTERNAL Asset
   ;

/*********************************************************************
      Configuration of Function and Security
 *********************************************************************/

/* The configuration parameters directly determine the configuration
 * of the system, and indirectly determine the initial assets that are
 * available.  There is not a 1-1, but a many-to-many, relationship
 * between parameters and assets.
 * */

ConfigurationDeclarations:
   | ConfigurationDeclarations ConfigurationDeclaration
   ;

/* Every configuration parameter may take on one or more values: all
 * possible values are listed in the declaration.  Represent both
 * hardware and software configuration by parameter settings.
 *
 * The value that a configuration parameter takes on, perhaps in
 * combination with the values of other parameters can affect the
 * availability of system assets.  The exact combinations that provide
 * assets will be noted with the assets.  In this production, simply
 * note which assets can be affected by the parameter.
 * */

ConfigurationDeclaration:
   ParameterDeclaration
   VALUES ":" DiscreteValues
   AFFECTS ASSETS AssetNames
   ;

ParameterDeclaration:
   ParameterKind PARAMETER ParameterName
   ;

ParameterKind:
   FUNCTIONAL
   | SECURITY
   ;

ParameterName:
```

```
   Identifier
   ;

/*********************************************************************
            Assets
 *********************************************************************/

/* Declare each asset available to the system, whether provided by the
 * external environment, configured with the system or generated by an
 * application.
 * */
AssetDeclarations:
   | AssetDeclarations AssetDeclaration
   ;

AssetDeclaration:
   AssetOrigin AssetType Asset ValuesRequirements
   ;

/* An AssetOrigin describes the manner in which the asset becomes
 * available.  It is a matter of static semantics that an EXTERNAL
 * asset be declared only in an EnvironmentDeclaration.
 * */
AssetOrigin:
   EXTERNAL
   | CONFIGURED
   | GENERATED
   ;

/* It may be useful to model more kinds of assets, but for now, we
 * have resources, which have associated quantities (e.g. 100 megs of
RAM),
 * capabilities (e.g., e-mail) which are either present or absent
(discrete
 * value 1 or 0) and features (e.g., mean time between failures). We can
 * be pretty abstract about assets, but each one should have a name and
 * a discrete value.
 * */
AssetType:
   RESOURCE
   | CAPABILITY
   | FEATURE
   ;

/* Assets are always quantifiable or enumerable.
 * */
Asset:
   Quantity OF AssetRights AssetName
   | DiscreteValue OF AssetName
   ;

AssetNames:
   | AssetNames AssetName
   ;

/* A flat name space of assets is assumed.  An improvement would to be
 * to have structured asset names.
```

```
 * */
AssetName:
   Identifier
   ;

/* AssetRights describe how an asset can be used.  A SHARED resourse
 * may be used by anyone without restriction.  An EXCLUSIVE asset may
 * be used by only one user at a time.  A DIVISIBLE asset may be
 * apportioned to users in chunks over which they have exclusive
 * rights.
 * */
AssetRights:
   SHARED
   | EXCLUSIVE
   | DIVISIBLE
   ;


/* Quantifiable assets are expressed as some number of asset units.
 * For examples:
 *
 *   100 Megabytes
 *
 *   28800 baud
 *
 * */
Quantity: Number Unit;

Unit: Identifier;

/* System assets are governed by combinations of parameter settings.
 * Each asset may have several possible associated values, each with
 * its own relationship to parameter settings.
 * */
ValuesRequirements:
   ValuesRequirement
   | ValuesRequirements ValuesRequirement
   ;

/* Each asset value may be brought about several ways through
 * parameter settings, each way described in a
 * ParameterSettingsDisjunct.
 * */
ValuesRequirement:
   VALUE DiscreteValue REQUIRES ParameterSettingsDisjuncts
   ;

ParameterSettingsDisjuncts:
   | ParameterSettingsDisjuncts OR ParameterSettings
   ;

/* Each way to bring about an asset value through parameter settings
 * is of the form
 *
 *   ParameterSettingSet AND ... AND ParameterSettingSet
 *
 * Each ParameterSettingSet is in one of the following forms
```

```
 *
 *    p in {v1, ..., vn}
 *
 *    p in [v1, v2]
 *
 *    p = v
 *
 *   The first two forms are governed by DiscreteValues.  The third
 *   form is a special case of each of the first two, given for
 *   convenience only.
 * */
ParameterSettings:
   | ParameterSettings AND ParameterSettingSet
   ;

ParameterSettingSet:
   ParameterSetting
   | ParameterName IS IN DiscreteValues
   ;

ParameterSetting:
   ParameterName HAS VALUE DiscreteValue
   ;

/*********************************************************************
              Requirements
 *********************************************************************/

/* The idea is to describe all the requirement sets that could be in
 * effect for the system.  There is no claim that any particular
 * requirement set is in effect.  Contrast this with
 * ApplicationDeclarations, in which there is exactly one requirement
 * set per mode.
 * */

/* The RequirementSets production is for the family of requirement
 * sets.  Each requirement set in the family corresponds to a
 * RequirementSet production.
 * */
RequirementSets:
   | RequirementSets RequirementSet
   ;

/* Each requirement set consists of two lists of statements, one list
 * describe asset conditions required by the system, the other list
 * asserting asset conditions provided by the system.
 * */
RequirementSet:
   REQUIRES AssetConditions
   PROVIDES AssetConditions
   ;

AssetConditions:
   AssetCondition
   | AssetConditions AssetCondition
   ;
```

```
/* At the atomic level, an asset condition is a statement of the
 * presence of an asset with a particular quantity or value, or
 * a relationship between different assets.
 * */

AssetCondition:
   Asset
   | Term Comparison Term
   ;

Comparison:
   "<"
   | "<="
   | "="
   | "<>"
   | ">="
   | ">"
   | IS IN
   | IS NOT IN
   | CONTAINS
   | DOES NOT CONTAIN
   | IS A SUBSET OF
   | IS A SUPERSET OF
   | INTERSECTS
   | DOES NOT INTERSECT
   ;

/*
 * An asset name can be used in a comparison expression to mean
 * the value of the asset. For example: Memory >= 5 MB.
 **/

Term:
   AssetName
   | Quantity
   | "{" DiscreteValues "}"
   ;


/*********************************************************************
              Applications
 *********************************************************************/

ApplicationDeclarations:
   | ApplicationDeclarations Application
   ;

/* The requirements stated with the application itself are distributed
 * over all the modes of the application.  This way, repetition is
 * avoided.
 * */
Application:
   APPLICATION ApplicationName
   UserRequirements
   AdminRequirements
   Modes;
```

```
ApplicationName: Identifier;

Modes:
   | Modes Mode
   ;

Mode:
   MODE ModeIdentifier UserRequirements AdminRequirements
   | MODE NULL
   ;

ModeName:
   NULL
   | ModeIdentifier
   ;

ModeIdentifier:
   Identifier
   ;

UserRequirements:
   USER REQUIREMENTS RequirementSet
   ;

AdminRequirements:
   ADMIN REQUIREMENTS RequirementSet
   ;

/*********************************************************************
              Integration
 *********************************************************************/

/* Because of interactions among applications, allocation functions
 * for resources are nonlinear.  For example, application Alpha may
 * require 25% of the CPU for adequate responsiveness when running by
 * itself, but three other application that each require 25%, Bravo,
 * Charlie, and Delta, could not run simultaneously with adequate
 * responsiveness.  There can be complex interactions among
 * configuration parameters, resource values, and application modes
 * that affect the allocation of a particular resource.
 *
 * The nonlinearity of allocation is far too complex to model exactly,
 * but useful approximations can be made by use of linear functions
 * within small regions.  We divide the space of independent variables
 * into multidimensional rectangular regions and give the linear
 * approximation function for each region.

 * */

/* Specify all integration modifications here.
 * */
IntegrationRequirements:
   INTEGRATION IntegrationList
   ;

IntegrationList:
   Integration
```

```
   | IntegrationList Integration
   ;

/* Separately specify the nonlinearities, asset by asset.
 * */
Integration:
   ASSET AssetName REQUIRES Approximations
   ;

/* Approximate the allocation function for the asset region by region.
 * */
Approximations:
   Approximation
   | Approximations Approximation
   ;

/* Each region for approximation is determined by a list of intervals
 * of values for the quantities that affect the allocation function.
 * */
Approximation:
   APPROXIMATION SLOPE Number INTERCEPT Number WHEN ValuesList
   ;

/* Specifiy the edges of the n-dimensional region of values.
 * */
ValuesList:
   Values
   | ValuesList AND Values
   ;

/* Values can be for configuration parameters, assets, or applications.
 * */
Values:
   ParameterSettingSet
   | AssetValues
   | ApplicationValues
   ;

AssetValues:
   ASSET AssetName IS IN DiscreteValues
   | ASSET AssetName HAS VALUE DiscreteValue
   ;

ApplicationValues:
   APPLICATION ApplicationName IS RUNNING
   | APPLICATION ApplicationName IS IN MODE ModeName
   ;

/*********************************************************************
              System State
 *********************************************************************/

/* The particular configuration and state of the system is input to
 * the tool via this production.
 * */

SystemInstance:
```

```
   CONFIGURATION  ConfigurationSettings
   ASSETS         AssetSettings
   APPLICATIONS   ApplicationSettings
   ;

ConfigurationSettings:
   | ConfigurationSettings ConfigurationSetting
   ;

ConfigurationSetting:
   ParameterKind PARAMETER ParameterName HAS VALUE DiscreteValue
   ;

AssetSettings:
   | AssetSettings AssetSetting

AssetSetting:
   ASSET AssetName HAS VALUE Quantity;

ApplicationSettings:
   | ApplicationSettings ApplicationSetting
   ;

ApplicationSetting:
   APPLICATION ApplicationName
   | APPLICATION ApplicationName IS IN MODE ModeName
   ;
```

6.4 Potential analysis support

The field of operations research has developed extensive tools and techniques to solve the problem of determining optimal or near optimal functionality possible with a given set of resources. These techniques require that the criteria used to evaluate alternative scenarios be as precise, rigorous and unambiguous as possible. If the security protections of a system have been compromised, the goal of maximizing functionality is not a sufficient criterion for evaluating alternative reconfigurations. Preventing the disclosure of sensitive data or a takeover of the system (or of some other system networked with it) can be just as important as maintaining a high level of functionality.

Thus, existing optimization techniques do not apply to our problem. A reconfiguration decision entails so many unquantifiable and intangible goals, that only a human commander can make it. The best formal techniques can do is to support such decision-making; and we believe that the formalisms, methods, and algorithms developed for operations research can be used to clarify a commander's situation in many ways—for example, by providing a level of abstraction at which much of the low-level system configuration details are hidden. Automated techniques may also be useful for pruning the solution space by eliminating impossible or infeasible solutions and for presenting some of the consequences of various reconfiguration choices. These consequences could be deduced both numerically and heuristically.

7. Tech Transfer

Given the current state of tool support for system configuration, we believe that the best course for this effort is to integrate our work at the level of a *model* that relates requirements and configuration parameters. This will at least pave the way for incorporating our results into actual tools. We have examined EMMA and SSAT. We believe our model is mostly compatible with their approach and do not see any major obstacles to integration at the model level.

To integrate (at the model level) with other DARPA projects or with COTS tools, it may be best to develop our model using the concepts and terminology of the Unified Modeling Language (UML). The UML diagrams that look most useful for our purposes are the Class Diagram and the Deployment Diagram (see, for example, [9], Chapter 10). Package diagrams may also be useful. UML diagrams provide strong support for recording dependencies among classes, packages, and components. They do not, however, specifically support the concepts of attributes or specifications that relate requirements on cyber assets to finer-grained requirements on sub-components. Since our model has so far been developed independently of the UML language, alternative representations are certainly possible.

8. Conclusions and Recommendations

8.1 Results

We have proposed a method for helping to make reconfiguration decisions. The method is neither top-down nor bottom-up, but starts in the middle by abstracting the capabilities of application modules into various modes of operation. We have examined three simple examples to indicate how the method might work, and to illustrate both its strengths and its weaknesses. We also sketched the beginnings of a modeling framework to help put the method on a systematic basis and to help in the development of support tools.

8.2 Recommendations

Hard work is needed to validate our approach. This work includes: constructing some basic tool support, so that the specification of the details does not become too cumbersome, and applying the method to a project developing a system that must be able to withstand a cyber attack. "Application" development should be part of the project, so that we could exert some control over the establishment of application modes and resource needs.

9. References

[1] McCullough, D., Korelsky, T., and White, M. Information Management for Release-based Software Evolution Using EMMA. in 10th International Conference on Software Engineering and Knowledge Engineering (SEKE '98). 1998

[2] Ascent Logic, *RDD-100 System Designer*. 1992.

[3] Vitech Corporation, *CORE User Reference Guide*. 1993-1996, Vienna, VA.

[4] Rosenthal, D., Hird, G., McCullough, D., and Thomas, R., *Final Report on Composability Methodology*. 1995, Odyssey Research Associates.

[5] Rosenthal, D., Samsel, P., and Barbasch, C., *Final Report: Information Tools for Security Protection*. 1997, Odyssey Research Associates.

[6] QSS Inc., *DOORS*. 1999.

[7] Rational, *RequisitePro*.

[8] StarBase, *StarTeam*. 2000.

[9] Fowler, M., UML Distilled: Applying the Standard Object Modeling Language. 1997: Addison-Wesley

[10] Object Management Group, CORBA Services: Common Object Services Specification, Chapter 15, Security Service Specification. Framingham, Massachusetts, Object Management Group, 1998.

*U.S. GOVERNMENT PRINTING OFFICE: 2001-610-055-10135

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.