RL-TR-93-49
Final Technical Report
May 1993

INTEGRATED TRUSTED SYSTEMS DEVELOPMENT ENVIRONMENT

ORA Corporation

Tanya Korelsky and David Rosenthal

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

APPROVED: Emilie J. Siarkiewicz

EMILIE J. SIARKIEWICZ
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control and Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE May 1993	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE INTEGRATED TRUSTED SYSTEMS DEVELOPMENT ENVIRONMENT		5. FUNDING NUMBERS C - F30602-91-C-0058 PE - 35167G PR - 1069 TA - 01 WU - P3
6. AUTHOR(S) Tanya Korelsky and David Rosenthal		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ORA Corporation 301 Dates Drive Ithaca NY 14850-1313		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3AB) 525 Brooks Road Griffiss AFB NY 13441-4505		10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-93-49
11. SUPPLEMENTARY NOTES RL Project Engineer: Emilie J. Siarkiewicz/C3AB/(315)330-3241.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document is the Final Report of the Integrated Trusted Systems Development Environment (ITSDE) project. The ITSDE project was a feasibility study that addressed the software development process for trusted systems. The goals of this effort were: (1) to elaborate the Integrated Development Process (IDP) for trusted systems outlined in "Developing Trusted Systems Using DoD-STD-2167A" by T. Benzel and to elaborate the corresponding documentation suite, and (2) to investigate how formal specification and verification tools developed at ORA for Rome Laboratory fit into the elaborated IDP for trusted systems of high assurance. The IDP attempts to integrate activities by the DoD-STD-2167A and the "Trusted Computer System Evaluation Criteria", TCSEC, into a unified approach for the development of trusted systems. We chose to investigate the elaboration of the IDP by developing requirements, design, and formal specification of a particular example. This approach allowed us to illustrate the findings and recommendations produced by the study. As our example for the study we chose to add a trusted mail service, which we call the Trusted Mail Handler, to the THETA (Trusted Heterogeneous Architecture) distributed operating system. THETA adds trusted distributed operating system functionality on top of commercial-off-the-shelf (COTS) trusted operating systems.

14. SUBJECT TERMS Software development formal specifications trusted systems			15. NUMBER OF PAGES 76
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L

Contents

List of Figures

Chapter 1

Executive Summary

This document is the Final Report of the Integrated Trusted Systems Development Environment (ITSDE) project conducted for Rome Laboratory by ORA. The ITSDE project was a feasibility study that addressed the software development process for trusted systems. The goals of this effort and the parallel effort at Trusted Information Systems were

- to elaborate the Integrated Development Process (IDP) for trusted systems outlined in "Developing Trusted Systems Using DOD-STD-2167A" by T. Benzel [Ben89] and to elaborate the corresponding documentation suite, and
- to investigate how formal specification and verification tools developed at ORA for Rome Laboratory fit into the elaborated IDP for trusted systems of high assurance.

The IDP attempts to integrate activities mandated by the DoD-STD-2167A [Dep88] and the "Trusted Computer System Evaluation Criteria", TCSEC [DoD85] into a unified approach for the development of trusted systems. We chose to investigate the elaboration of the IDP by developing requirements, design, and formal specification of a particular example, according to the first version of the IDP [Ben89]. This approach allowed us to show various inadequacies of the first version of the IDP process and to illustrate the findings and recommendations produced by the study. As our example for the study we chose to add a trusted mail service, which we call the *Trusted Mail Handler*, to the THETA (Trusted HETerogeneous

Architecure [RL992]) distributed operating system. THETA adds trusted distributed operating system functionality on top of commercial-off-the-shelf (COTS) trusted operating systems. It is an object-oriented system, which supports the addition of new trusted services.

Our feasibility study showed that the original version of the IDP that was developed for systems is not completely suitable for trusted applications and needs modifications and additions. One of the results of the study showed that there should be variants of the IDP corresponding to a variety of factors such as the software type (e.g., system vs. application) and level of trust. We call a particular development process based on the development factors a *thread* of development. We introduce the concept of an IDP with multiple threads in Chapter 2. In Chapters 3, 4, and 5 we present a general discussion of the needed changes in the IDP process and the related changes in the corresponding IDP documentation.

The second direction of the study was to investigate how the Romulus security methodology fits into the elaborated and modified IDP. The results of the study showed that the Romulus methodology can be used in several ways in the integrated development process, ranging from the conventional design specification "from scratch" to the reuse or adaptation of Romulus models available in the library of models. We present the discussion and the results of this second direction of the study in Chapter 4.

We also discuss how the THETA development methods are part of a particular thread of development (Chapter 5). The standard THETA development thread requires substantially less security analysis then an analysis of an OS and provides insight into the utility of having different development threads for different kinds of trusted development.

In our proposal for the ITSDE project, we anticipated that our study of the issues described above would allow us to discover deficiencies and recommend improvements to the systems under investigation, i.e., Romulus and THETA. Our observations are presented in Chapters 4 and 5.

In Chapter 6 we present a sketch of how the ITSDE design could be incorporated into the Software Life Cycle Support Environment (SLCSE). SLCSE fully supports the DoD-STD-2167A software life cycle and thus is particularly well suited for supporting ITSDE. The advantages of providing software support for the ITSDE concept are particularly significant because of the complexity of interconnecting common development factors in defining a thread.

Chapter 2

ITSDE: Multiple IDP's

2.1 Introduction

The formulation of the Integrated Development Process (IDP) by T. Benzel in [Ben89] was concerned with integrating the TCSEC documentation requirements into the DoD-STD-2167A software development process. However, this formulation essentially outlined only one development and evaluation process. Different kinds of trusted systems require different kinds of development and evaluation processes because systems differ in the kinds of security services they provide and in their external interfaces to other trusted components. This fact was recognized by the National Computer Security Center (NCSC) in their formulation of network evaluation guidelines which was published as the Trusted Network Interpretation (TNI)[NCS87] in 1987. The Trusted Database Interpretation (TDI)[NCS91], which appeared in 1991, continued the diversification of the development and evaluation process to applications. D. Bodeau in [Bod88] has described many of the issues involved in integrating the development of applications into DoD-STD-2167A, including the difficulties of building trusted applications. In this report, we expand on the application development process, with particular attention paid to the incorporation of high assurance methods and tools.

To produce a detailed explanation of the development process, we must take into account that systems that require different degrees of TCSEC trust (e.g., C2, B3, A1) have different requirements for secure development. For example, in producing a top level design, we must choose between assurance methods such as disciplined software engineering, manual formal methods, and mechanical formal tools.

In addition to the kind of system and level of trust there are other factors that are needed for determining the correct development process. Based on all of the factors one can define particular development methods that we call *threads* of the integrated development process. We will address the factors in choosing a thread in Section 2.3.1.

We call the environment to support the modified IDP based on multiple threads the Integrated Trusted Software Development Environment (ITSDE). In this chapter, we examine what is needed for thread development.

2.2 The 2167A Phases

Before beginning the thread descriptions, we give an overview of how to incorporate security into the 2167A development process. For a substantially more detailed exposition, see D. Bodeau's report [Bod88].

The 2167A development process is often called a "waterfall" model, as at the completion of one phase of the process one drops into the next phase. This method is best suited to handling projects that are well understood so that one can correctly plan out the project without knowing all of the details. Projects that involve more research need a way to reevaluate and reformulate what needs to be done when an unexpected difficulty arises. These projects are often handled by a spiral method [Boe88]. In this report we concentrate on the traditional 2167A approach. For a discussion of ways of handling the spiral method, see [TIS89].

Because many of the important security decisions are made before the 2167A process begins, it is desirable to augment our discussion of the 2167A development phases with an earlier phase. We will call this the early requirements phase, as its purpose is to produce a first draft of what the system or application should be like (typically in the form of a proposal).

Here is a short description of security aspects of the the combined early requirements and 2167A phases.

- Early Requirements: In this phase the general method of developing the project is chosen, including the initial identification of the potential security threads that may be used.

- Requirements: The requirements for the system and its components are identified. In particular, the security requirements are identified.
- Top Level Design: The specification and development process decisions are made more concrete. This phase includes identification of trusted objects and how they will be handled. It sometimes includes an initial examination of potential covert channel problems. A top level specification may be produced.
- Detailed Design: All of the design details are completely developed. This phase includes checking the correct handling of trusted objects. A more detailed description of the security aspects of the design is typically produced at this stage.
- Code: The system or application is implemented at this stage. A code-design correspondence is made to assure that the code follows the design. (This correspondence includes checking that the code follows the security model.) Some covert channel problems may be discovered and resolved at this stage.
- Integration and Testing: The parts of the system are put together and tested. Interactions between parts of the total system, may impact on the overall security (e.g., covert channel problems) and must be tested. Penetration testing is also needed.

We present more details about the early requirements phase in Chapter 3, the requirements and top level design stages in Chapter 4, and the detailed design and code in Chapter 5.

2.3 What Goes into a Thread

Each supported thread should include information for all the phases of the DoD-STD-2167A (including the early requirements phase). This information will include guidance documents, development methods and tools, and examples. Before describing these parts, we first give a general description of factors involved in defining threads.

2.3.1 Thread Definitions

The details of particular thread construction are beyond the scope of this effort, but to provide some insight into what is needed, we give a short sketch of some of the factors that should be taken into account for defining a thread:

- the security assurance level
- the security functionality that the system provides
- the security support services that are required of the underlying platform
- the constraints imposed by meeting other kinds of objectives such as real-time constraints
- the choice of development method (for example, the THETA autogeneration method described in Chapter 5)
- specialized factors such as
  - special hardware or operating system characteristics
  - security portability of the application or system being developed
  - security concerns arising from distributed computing

In its full generality, building threads from factors looks like a formidable task. However, much of the simplification and grouping of factors has been done in the construction of existing guidelines (see Section 2.3.2.1). Also, ITSDE can grow incrementally, starting with several better understood threads and adding more threads gradually. The complexity of the thread definitions can be managed by using the software support of the ITSDE/SLCSE system (see Chapter 6).

2.3.2 Documentation

A thread definition involves more than just a classification of what is needed in the development process. There should be documents and tools associated with a particular thread. In this section and the following section (2.3.3) we examine these aspects of threads.

2.3.2.1 Software Documentation

For each thread, a detailed description is needed of the variants of the 2167A Data Item Descriptions (DIDs) tailored to the thread.

To define the collection of documents that should be produced for trusted systems in ITSDE and the variants of these documents tailored for particular threads, one should build on the results of work done by

- the NCSC, such as the NCSC Rainbow Series
- the group at the AFCRC (the Cryptological Research Center at the Kelly Air Force Base), such as the Guide for Security Relevant Acquisitions CDRL and DID Handbook
- TIS, who produced some guidance for tailoring a general-purpose operatings system and two stand-alone DIDs (for the Security Policy and for the Philosophy of Protection), and recommendations for tailoring the existing 2167A DIDs in their Final Report for the ITSDE project [BR93].
- Deb Bodeau in her report "Guidance for Reviewers of Security-Relevant Design Specification and Verification Documentation" [Bod88]
- MITRE in their SCAP Workbench tool
- Jody Frosher and Charles Payne in their "Navy Handbook for Trusted Application Systems"
- ORA in this report
- other groups within DoD defining acquisition and certification CDRLs and DIDs

One should also take into consideration the currently emerging Federal Criteria, the first draft of which should be available in the fall of 1992. Another relevant guideline being prepared is the "Guideline on Developing Security-Related Mission Needs".

2.3.2.2 Descriptions of COTS Products

Threads will be partly based on the security services that the composite system needs from the underlying COTS products. Especially important are any details needed to make the integration secure. Consequently, detailed documentation of thread-specific trusted COTS products evaluated or accredited by the Air Force should be available to the ITSDE user.

2.3.2.3 Supporting Guidelines Material

For each thread, the collection of relevant mandated government documents related to particular development phases should be available to the user. Additional documents may be needed where the government documents do not provide enough guidance, for example, documentation for trusted applications. [1]

2.3.3 Development Methods and Tools

Tools supporting assurance methods should be integrated into ITSDE and be available for the user depending on the type of development thread. In Chapter 4 we discuss the Romulus tool. Other tools, such as verification tools (Gypsy, EHDM, and FDM) should also be included.

2.3.4 Libraries and Reuse

It is highly desirable to make the trusted development as routine as possible by having threads contain the security analysis that is common to all uses of that thread. The nature of the reuse of existing examples depends on the complexity of the security needed by the system or application. Where it is simple, there is likely to be a worked out method that could be tailored or instantiated to the specific circumstances. Where it is complex, one may have to reason by analogy with an existing application.

Some of the development threads may have pre-analyzed formal models (for example, in the Romulus library of models) that can be used in the

[1] Currently, there are several working groups addressing this and related problems in the DoD. An example of new documentation of this kind is the "Navy Handbook for Certification of Trusted Application Systems", which will be presented at the MILCOM '92 in October.

requirements/design stage. There also may be sophisticated code generation, or code-model correspondence establishment techniques. Consequently, for threads corresponding to the higher levels of trust, ITSDE should include the library of supported models (both reusable and adaptable) and corresponding reusable methods of design and code generation.

2.4 Certification

In the latest draft documents produced under the Federal Criteria Project, such as the "Minimum Security Functionality Requirements for Multi-User Operating Systems" [NIS92], the NIST and NSA outline the government plans to "broaden the trusted product evaluation program substantially by accrediting laboratories to evaluate commercially-oriented products", [NIS92], pp. 1-6. It means that the DoD services will be authorized to evaluate/certify a full spectrum of trusted products and systems.

ITSDE should fully support the evaluation/certification/accreditation process. By certifying threads as well as particular systems and applications, much of the complexity of the security analysis can be removed from the actual software development.

Recently developed tools like the SCAP (Security Certification and Accreditation Plan) Workbench Tool describe in detail the certification activities that should happen during the software development stages. This tool, however, stands alone and is not currently integrated with the DoD-STD-2167A process. For example, the certification reviews that are mandated by the NCSC are not correlated with the suite of reviews mandated by the DoD-STD-1521B, the review standard that is used in the DoD-STD-2167A paradigm. We believe that the certification/accreditation process should be supported by ITSDE in an integrated fashion. The close monitoring of the trusted system development by the Designated Accreditation Authority (DAA) via ITSDE will help ensure that all the mandated and necessary development is performed and documentation is produced in a timely manner. It will also streamline the overall evaluation/certification/accreditation effort considerably.

The development and certification of a thread is going to be difficult and will require considerable expertise. The certifiers of a thread will need to be experts in security theory and practice. The certifiers need not be the same organization responsible for checking that a system or application conforms to an existing approved thread.

2.5 Development of a New Thread

In this section we examine, what is involved in developing a thread (as opposed to what is involved in developing a system or application). There is a considerable difference between what one should do for following a thread in developing a system or application and what is involved in the development of the thread. In this section we sketch what is involved in developing a thread.

Because of the cost, one should strive to make a thread general so it can be applied to a relatively large class of systems or applications. In addition, the thread definition should be clear to both software developers and security software certifiers.

Here is a sketch of a thread development method:

1. identification of the security needs of the application (or class of applications)
2. analysis of adapting existing techniques or the analysis of what is needed in a new security methodology
3. generalization of the step 2 work to handle a larger class of problems
4. development of the security method, taking into account both design and coding issues.
5. analysis and possible verification of part of the method.
6. *documentation production*
7. certification of methodology process
8. adaptation of ITSDE/SLCSE (incorporation into a library of methods)

Chapter 3

The IDP for Trusted Applications: The Early Requirements Phase

3.1 Trusted Applications

We use the term *application* software to mean the additional software added on top of some existing computer system. In particular we assume that the operating system and possibly other supporting software are already built. The word *trusted* is used in the security literature in many different senses. In this report, a "trusted application" will mean an application that handles information of multiple security levels and is trusted to do so in a secure way.

The security concerns associated with developing an application are distinct from those of an operating system. Typically, the application need not duplicate many of the operating system security services, such as authentication. Hence the security analysis of the application will depend on the security services provided by the underlying system. Additional security services provided by the application may depend on partial correctness of some parts of the underlying system.

The overall security provided by trusted parts of an application may rely on characteristics of the underlying system that may be difficult to analyze (such as covert channel interaction). Therefore, the simplest and safest course of action is to minimize the trusted part of an application. Indeed, the current security guidelines for applications (i.e., TDI) discourage the use of any applications handling multilevel information. Because of the TCB minimization requirement of TCSEC B3-A1 class systems, such applications would not be considered for B3 and above. However, there is still active interest in developing trusted applications, particularly at the B1 level because of the added efficiency and functionality they have over equivalent applications built using only single-level processes.

3.2 Early Requirements Phase

Since there is no standard for the early requirements phase, we include one way in which this phase could be accomplished.

1. Identification of what is desired (goals) and what is needed. We refer to this step as the determination of the customer goals and customer needs. This includes determining what security measures are needed and the overall level of security assurance. A distinction between goals and needs is made so that the appropriate tradeoffs of requirements can be made.

2. Identification of what will be accomplished.

    - Identification of what the application will be. A discussion of the security tradeoffs will be needed where the customer goals cannot be met.
    - Identification of the thread of development. The general method for building the application is presented at this stage. This includes how the security objectives will be achieved and how unanticipated difficulties might be handled. Where possible one would utilize an existing approved development thread for achieving security. A rationale for the appropriateness of the method to the particular application is also given here.
    - Initial security analysis. An argument should be provided describing how the security of the composite system can be achieved by the functionality of the application and the underlying system.

This method is very much like the development of a proposal in response to a customer's request for proposal (RFP). Note however, that the best way to do the first step is to directly confer with the customer. In practice, this may mean that some aspects of the early requirements phase may have to be modified or expanded at the start of a contract. Also, the steps of this approach may need to be iterated until the proposed solution is acceptable to the customer.

Also, documentation describing what is needed for different possible ITSDE threads will need to be available to the contractors, probably before the award of a contract.

Considerations in early requirements development Before a security methodology is chosen it should be made clear to what extent it can be used to assure the attainment of some security goals. For example, formal security analysis with the restrictiveness theory generally does not examine potential timing channels. If the goal is to absolutely prevent any leaks, then this method of analysis will provide some but not complete assurance.

Other considerations include the following:

- What security interface conditions are appropriate/ To what extent are the security assumptions about the interfaces to the underlying system met?
- How does the operating environment of the application impact on the potential exploitation of channels. To what extent will this application increase the actual utilization of the systems covert channels?

3.3 THETA Mail Handler Example

We applied our integrated development process methods to a particular example, the THETA Mail Handler. THETA is a secure extensible distributed operating system that supports the development of trusted applications [RL992]. The application was the addition of a mail handler service on top of THETA. Our goal was not to build a system extension to THETA, but rather to investigate the development process. As such, we focused on those parts of the application development that would provide insight into the development process.

The functionality of mail handling consists of

- **auser interface** that displays options and unread mail and handles composition of messages.
- a **mail manager** that keeps track of arriving mail, retrieves a message to be read, and other operations.
- a **notifier** that notifies a user that there is unread mail.

The user interface will operate in a window at a single level. It can be untrusted.

In this chapter, we examine the early requirements phases, in the next two chapters, we will also discuss this example for later stages of development.

3.4 Early Requirements Phase for the THETA Mail Handler

The first part of this phase was the eliciting of customer goals and needs. A document describing the results is contained in Appendix A.

Two key security characteristics were:

- Covert channels were to be eliminated where feasible.
- The application was to be MLS if this would aid efficiency.

In developing the proposed solution several alternatives were explored. Ideally the different alternatives should already be well documented, existing threads, so that one does not have to explore what are the ramifications of the different choices. Since at the time, there were no documented threads, our development path was forced into more of a research direction.

The options we explored included:

- No MLS managers (i.e., use a collection of single level processes)
- A standard THETA Manager development approach for the Post Office and User Interface

- The use of a "THETA Application" instead of a THETA Manager. (There is more than one way in THETA to set up an application. The distinction is primarily not a security concern.)

- A "restrictive" THETA Manager (with sharing of a common mail datastructure for mail at different levels)

Efficiency benefits and security problems were surfaced for the MLS approaches. This involved developing an understanding of the capabilities of THETA Managers. Utilizing information about the current THETA system security (which would be contained in ITSDE) we noted the security channel due to the limitations of the secure AT&T UNIX COS to recognize requests parameterized by security level.

An "exploratory design" of the restrictive THETA Mail Handler was performed (a spiral approach), to gain a better insight into potential design or coding problems.

Chapter 4

Requirements and Top Level Design Phases

In this chapter, we discuss the requirements and early design phases of application development. Our main focus is on how to integrate the Romulus security modeling tool, into these phases of the development process. We assume that the early requirements phase has been completed so that the proposed solution and general method for handling the development have already been chosen.

4.1 Romulus

Romulus [ORA90] is a modeling tool that supports both

- a top level data flow decomposition tool that checks security constraints between components, and
- a specification language, security condition generation mechanism, and verification tool for checking the security of a specification. Romulus provides the most support for showing that a formal top level specification (FTLS) satisfies the security property known as restrictiveness [McC90], although it could be used to check other security properties.

4.2 Placement of the Romulus Activities in the IDP

4.2.1 Problem: Requirements/Top Level Design Dilemma

One of the difficulties of integrating the security development process with 2167A is deciding where to place the security modeling. One would expect to produce the security requirements during the requirements phases (SSS to SRS). A natural step in the requirements formulation is the construction of a security policy model. The security policy model, however, requires the identification of subjects and objects, and the identification of objects is an internal detail of a component and really part of a design decision (SDD).

This problem is not unique to security, but rather due to the fact that some software properties are most easily expressed and understood by relating them to design or even implementation level concerns. One could require a pure "black box" development approach and not allow the expression of top level requirements based on details of the internal state. Indeed, trace based security models are intended to do this. We believe, however, that the advantages to identifying, at least in general way, what objects need to be protected outweigh the disadvantages of prematurely constraining the design.

4.2.2 Solution: Suggested Way of Handling Placement of the Modeling

We believe the solution to the problem is to put the modeling into both the requirements and the top level design phases.

4.2.2.1 Requirements Phase

In the requirements phase, the suggested model and method from the early requirements phase is made more precise. This activity includes the following

- decomposition of the top level components
- the identification of the trusted components
- the dependencies of these components (including data flow)

- the allocation of any special security requirements
- a preliminary indication of what the trusted components must protect and what security services they must perform
- identification and development of the security model
- an initial examination of any security flaws

Note that some of the items normally call for at least some general discussion of the objects being protected and of internal interfaces, but the discussion does not need to be detailed (as at the level of the SDD).

The Romulus system can be used to construct the components and their connections, to check that the interface connections between the components are consistent, and to surface security violations (such as sending secret information to an unclassified process or user). The graphical specification can later be filled in with more details to become part of the interface and design documentation.

4.2.2.2 Design Phase

In this phase the model is filled in and the formal and descriptive top level specifications (FTLS and DTLS) are produced. They are called specifications, but in practice they are really top level design documents and go into the design phase of 2167A.

Our approach differs somewhat from Benzel's IDP in that the FTLS and DTLS are not necessarily split into different phases of the 2167A process. The top level design needs a descriptive exposition, and if more assurance is needed, a formal exposition is also needed. The detailed design phase also needs a descriptive exposition. In practice, because of the cost, there would generally not be a detailed FTLS for the detailed design phase. However, critical parts of the design might benefit from a detailed formal description.

The textual specification language part of Romulus can be used to build a formal specification that can serve as part of the design documentation, and it can be used for verifying the security of that design (usually restrictiveness). This may involve writing a new specification or the instantiation of a more generic description. (Note that the kind of verification needed for instantiation will be different from a direct proof of security: for example, it may involve access control checks or checking special assumptions about the kinds of inputs or outputs.)

4.3 Kinds of Application Security Models

The choice of security model is primarily based on the kind of data sharing between processes at different levels and what kinds of potential security problems are intended to be surfaced. Here we describe a partition of the methods to highlight what is needed for the THETA Mail Handler.

4.3.1 Multiple Single Level (MSL) processes

One possibility is to prohibit any sharing of data across levels, i.e., the application is a collection of independent single level activities (e.g., processes). This model is what the THETA documentation calls MSL. In this case, there are no MAC issues, but there may still be development threads for this model to handle other security constraints.

4.3.2 Weak MLS applications

Another possibility is to allow only the scheduling activity and the event buffering to be shared. The application is conceptually equivalent to a collection of single level applications that are scheduled securely. The application is trusted to keep the data spaces separated. This is the primary model used for the THETA system managers and, in particular, is the model for the THETA Mail Handler. The term "weak MLS" is used because it is MLS in a very minimal way. There is almost no sharing of state information between the different activities.

4.3.3 MLS (or Strongly MLS)

In this case, each request that the application can handle may involve read-downs (accessing application information lower than that request) and may involve write-ups (altering information at a higher level). Without an adequate design, there is the potential that higher level information may be directly or indirectly placed into lower level objects.

There are a variety of formal models that are used to express MLS models, depending on what kind of information flows that are being analyzed. The standard approach is to use a Bell-LaPadula model, to make sure that there are no overt channels, e.g., direct access violations. Romulus utilizes a more thorough method, restrictiveness [McC90] (which is a variant of the Goguen-Meseguer non-interference method [GM82]). Restrictiveness identifies more potential security design problems.

4.4 Theta Mail Handler Example

In this section we continue the description of the THETA Mail Handler example for the Requirements and Top Level Design Phases. (Part of an adapted draft SSS was produced, see appendix D.)

The application was partitioned into three CSCs: the Post Office, the User Interface, and the Notifier (for incoming mail). The allocation of the MAC security requirements to these CSCs is straight forward. The User Interface was chosen to be single level and the Post Office was chosen to be a standard MLS THETA manager. The Notifier is an MLS process spawned as part of the "ideal" login process (which is planned for, but not yet implemented in the current version of THETA).

4.4.1 Romulus Specification

In the THETA development thread there is no need for an application developer to build a requirements/top level specification (FTLS) because the same model is used for all managers (weak MLS). We show how Romulus can be used to construct the model in Appendix C.

4.5 Suggested Modifications to Romulus

Based on our experience with modeling and specification here are our recommendations for modifications to Romulus.

- The current version of the specification language is not well suited for presentation to a broad audience. Either the language should be extended or a front-end language should be built.

- Inadequate support for specification of shared memory and scheduling is a limitation. We were not able to "naturally" specify the Theta manager method in Romulus on a detailed enough level. However, this deficiency is already being addressed by the Romulus developers (see [Sut92]).

Chapter 5

Detailed Design and Coding Phases

After the top level design has been produced it must be refined into a detailed design and then implemented. Typically we would expect that a DTLS would be produced at a detailed level and that it would be analyzed for its conformance to the top level design. For security critical parts of the code, a refined FTLS may also be produced. These would then be utilized in making a code-model correspondence.

Our principal focus in this chapter is on methods used by the THETA distributed operating system for the detailed design and coding of secure managers.

5.1 THETA Managers and the Security of the THETA method

The THETA method of building applications is one example of a thread of development. We first provide a brief explanation of the method of building THETA managers. The material in this section is a slightly modified form of part of the Theta Final Report [RL992].

> Most THETA managers are produced by a method called*autogeneration*. With this method, the basic properties of manager, its types, and the operations of each type are specified in a high-level language,

code for each operation is written, and the manager is assembled according to the specifications. The assembly involves using code from a standard *manager skeleton*, reusing the code of operations *inherited* from other types, and inserting the code for new operations.

A multi-level object manager is one that can handle operations on types that it manages at a range of security levels. A multi-level manager may be designed as a single multi-level secure (MLS) manager process or multiple single-level (MSL) manager processes. If it is implemented as a MLS process, then the manager is part of the mandatory TCB and is trusted to perform mandatory access checks.

THETA provides the programmer with a set of tools for manager generation. The programmer defines a type and a manager for it using the type-definition and manager definition languages. He then uses manager generation tools to build a skeleton of an object manager. The skeleton implements message packing and unpacking, conversion from canonical to internal representations of data and vice-versa, mandatory and discretionary access checks that may be necessary for an operation, and many other routines common to most managers. To finish implementing a manager, the programmer has only to fill in code for the type specific operations that the manager supports.

5.1.1 Security Issues in Autogeneration

The address space that a manager executes in is not partitioned by security level. If managers are single level (or implemented by the multisingle level (MSL) /refmsl scheme), then the single address space poses no concern. For MLS managers, however, care must be exercised in design and implementation so as to avoid any information flow that would compromise security.

Under the assumption that only the good guys write MLS managers, the threat of intentional malicious code is removed. But there could be oversights and inadvertent security violations. THETA manager generation tools provide the infrastructure to organize trusted manager development, and ensure that all of the basic security concerns are addressed.

5.2 Thread of Development and the THETA Mail Handler Example

Each standard THETA manager uses the same weak MLS model. An application developer is faced with no new security designs or analysis. There is, however, the potential that the code will not follow the model, so that a code-model correspondence is important. (One must still rely on checking the code to make sure memory is handled appropriately.) THETA provides documented guidance in achieving this aim.

Since most of the security work is done prior to the application development, this thread is particularly simple. This kind of situation is an example of the advantages of having specialized threads, and, in particular, of the THETA development approach.

THETA Mail Handler The detailed design and coding of the THETA Mail Handler should follow the THETA conventions developed for the construction of MLS managers. As this thread is already well developed, a detailed design and code for the THETA Mail Handler was not developed.

5.3 Recommendations for Theta

Suggestions for future upgrades to THETA include the following

- A simpler development environment that provides a better abstract interface to the system would be very useful

- Improved documentation would be helpful. Documentation for THETA manger development has already substantially improved during the ITSDE project [Cor92]. A detailed formal assurance argument needs to be written for a generic THETA manager

5.3.1 Modification to the Current THETA Method

An alternative to the current THETA method is to provide more automated assurance for (weak) MLS managers.

Here is one possibility that we and THETA project members have investigated. Any particular single level part of the process would have a separate address space (and be an OS lightweight thread of the manager). In order to achieve this, the first action of the lightweight thread would be to eliminate any potential references to the trusted part of the process (e.g., the queue of tasks). This action would be automatically generated as part of the autogeneration method. Under this scheme, we would not have to check that the application developer's code does not subvert the security of the process, but only that the autogeneration method was properly applied.

5.4 Code Verification and Penelope

The Penelope [Gua89] system is a tool for writing Ada specifications and code and verifying that the code meets the specification. As part of the effort on this project we have briefly investigated how Penelope would fit into the IDP. The example we have studied, the THETA Mail Handler, is designed to be an extension of THETA and hence would be coded in the C language. Therefore, Penelope is not directly applicable to this example. Below, we examine a few issues that should be considered in incorporating Penelope into an IDP thread.

5.4.1 Penelope Usage in an IDP Thread

Penelope could be used to discharge the assumptions from the FTLS and hence be part of the model-code correspondence. There may be specific constraints surfaced in the security design that could be checked in Penelope to provide a higher assurance that the code properly follows the design. One problem with adding this to a thread is the cost involved in code verification.

5.4.2 Modifications to Penelope

Here are two areas where modifications to Penelope would be useful.

- In the current system, it is awkward to prove the security property directly on the code since there are limitations in the analysis of input/output sequences.
- Currently, there is no connection between Penelope and Romulus and assumptions from the Romulus specifications must be manually translated into the Penelope specification language to be discharged in the code.

Chapter 6

SLCSE-based ITSDE Design Sketch

In this chapter, we briefly describe the Software Life Cycle Support Environment (SLCSE) [Str90] which was developed at Rome Laboratory's Software Engineering Laboratory to support DoD-STD-2167A development. We argue that SLCSE can provide a convenient platform for the implementation of the thread-based ITSDE and illustrate this argument with a high level sketch of examples of such an implementation.

6.1 The SLCSE system

SLCSE is an integrated collection of software tools and documentation that supports the 2167A development process. It includes tools for both the development and the management of software. SLCSE is not just a particular CASE tool, or a particular kind of software development environment, but rather an *environment framework.* This framework can be used to create a particular environment to suit the needs of a variety of different users on a software development project. It provides a "common consistent user interface accessing a comprehensive set of software development tools which support the full spectrum of DoD-STD-2167A software life cycle activities from Requirements Analysis to Maintenance" (p1-2 from [Str90]). It allows users assuming different roles (such as System Analyst, Project Manager, or Programmer) to have access to the tools and the documents that they need to carry out their roles.

The environment framework is supported by an underlying database, which maintains information relevant to the actual software and software documentation under development, and management information such as schedule and milestones. The results of user activities are interrelated through the database and user-interface (using hypertext-type links).

The database underlying SLCSE is an entity-relationship (E-R) database. Its collection of entities and relationships are subdivided into *schemas* corresponding to particular user roles. This way a user in a particular role has a more focused view of the tools and documentation supported by SLCSE.

For details on the SLCSE system see the SLCSE Final Technical Report [Str90].

6.1.1 E-SLCSE

The examples given in this chapter are mostly based on the existing SLCSE system. However, an enhanced version of SLCSE called E-SLCSE is currently being developed. The planned enhancements do not change the SLCSE concept, but instead provide a greater generality, flexibility and commercial quality to the environment. For example, the current SLCSE uses an Entity-Relationship (E-R) database; in E-SLCSE it will be replaced by a more general, object-oriented repository. This repository will be based on the PCTE repository.structuring the collection of entities (objects) and relationships by allowing sub-objects that inherit relationships from their parent object.

6.2 ITSDE/SLCSE Match

SLCSE can serve as a convenient platform for the ITSDE implementation because it provides a match for at least three concepts underlying ITSDE.

6.2.1 Thread Definition

A substantial enhancement in E-SLCSE over SLCSE is that it will support not only the 2167A process but other processes. In particular, E-SLCSE will support a process definition tool that will allow users to define their own variants of the software development process. This enhancement is very important for ITSDE because it will provide a convenient way to define threads.

6.2.2 Developer – Contractor Coordination

SLCSE can serve as a common medium between the developer/contractor and the DoD acquisition authority. By having access to the SLCSE database, in the Project Administration role, the acquisition authority has an opportunity to closely monitor the on going effort since the SLCSE database records in a well-organized form all the details of system development.

For trusted systems, there is another government entity, the DAA, that is mandated to closely monitor the system development from the certification/accreditation point of view. By adding a new role Certification Administration, SLCSE can provide a unique match for the ITSDE goals in that it can allow for the convenient access to and exchange of system development information between all the three parties involved via the SLCSE database/repository.

6.2.3 Production of Documentation

Another feature of SLCSE which (see Chapter 2) that accommodate the information mandated by TCSEC for the evaluation/certification process. The production of tailored documents can be supported in SLCSE in an efficient and convenient way, by using the specification language of the document generation tool. Material, which is stored in the SLCSE database, can be incorporated in more than one document, avoiding inefficient and time-consuming duplication. The E-SLCSE document generation tool will also provide consistency maintenance between the documents and the repository artifacts that were used in them, propagating changes in both directions.

Currently SLCSE uses a program called DOCGEN, which produces documents in LaTex. E-SLCSE will use more convenient, commercially available What-you-see-is-what-you-get document generation tools. The DOCGEN program (and its successor in the E-SLCSE) allows the user to specify the document structure in a specification language which, for every section of a document, points to the SLCSE database entity (or entity's attribute) containing material to be put in this section.

6.3 Implementation of ITSDE in SLCSE

In what follows we give an outline of a design and implementation process for incorporating ITSDE into SLCSE. The goal of this outline is to provide a general description of our approach and to show that the ITSDE/SLCSE configuration has significant potential and deserves future consideration

The first part of incorporating ITSDE into SLCSE is to produce a c‹ - plete design of the ITSDE development threads as outlined in Chapter 2.

After this part has been completed, the design and implementation on SLCSE should be laid out. The attractive feature of the ITSDE/SLCSE configuration is that no new software needs to be written except for the interfaces of the specification and verification tools (such as Romulus, Penelope, and other tools) with the SLCSE user interface and its database.

The design and implementation of SLCSE/ITSDE will include

- defining new SLCSE (or E-SLCSE) user roles,
- defining new database schemas as well as augmenting and changing the existing schemas,
- integrating the specification/verification tools into SLCSE, which involves
  - implementing the support for their invocation via the SLCSE user interface,
  - associating them with a certain (new) role in SLCSE,
  - connecting their input and output with particular objects in the SLCSE subschemas (new and existing ones),
- creating new document generation specifications for non-integratable security documents such as Philosophy of Protection [BBC90] and modifying the existing document specifications for 2167A documents according to the tailoring recommendations.

Full design of all these additions and modifications is clearly beyond the limits of the current effort. However, we believe that full design and implementation can be produced with a relatively modest effort after ITSDE threads are fully defined and configured. We next give an outline of this process, and provide illustrative examples of typical additions and changes.

6.3.1 ITSDE Thread Development

The design method for the ITSDE implementation on SLCSE will include a meticulous "walk-through" of all the threads designed for (the first version of) ITSDE (see Chapter 2). The walk-through will identify all the necessary additions and modifications to SLCSE. In particular,

- For each thread, all the additions and modifications to the SLCSE database/repository that are necessary to accommodate all the material and tools that are used in the thread should be identified and carried out. If a thread involves the use of tools not supported by SLCSE, these tools should be integrated into SLCSE
- For each thread, specifications in the SLCSE document generation tool of all the DIDs tailored to the thread (see discussion below in section 2) should be produced and tested
- As we pointed out above, the first version of SLCSE did not support explicit process definition. E-SLCSE, however, will support this feature and its user interface most probably will be able to support the choice and navigation of threads. It remains to be determined how much additional effort to support the ITSDE user interface will be needed

6.3.2 Examples of SLCSE Modification for ITSDE

Below we give examples of the SLCSE modifications necessary to support ITSDE. We give the examples using SLCSE terminology but all of them can be directly translated into the E-SLCSE terminology. The examples below are by no means are an exhaustive list of additions and modifications.

6.3.2.1 New Roles

The tools and documents available to a SLCSE user are configured by roles. Currently SLCSE supports eighteen roles that can be grouped as follows:

- different aspects of government contract acquisition such as Acquisition Management, Project Administration, Project Management

- different phases of system development such as System Analysis, Software Analysis, etc.
- SLCSE-specific aspects such as SLCSE Installation
- general roles such as Secretarial

For ITSDE, a role of Certification/Accreditation Management should be added. More analysis of the certification/accreditation activities needs to be done to decide whether it should be one role or a small collection of roles. For example, the "penetration team" role can be seen as related to such existing roles as Verification and Validation and Software Testing. However, it might be different and specific enough (for instance, might use specific tools and methods) to be assigned a special new role Penetration Study.

Another possible new role might be described as a Security Analyst. For systems at a higher level of trust, there will be security specialists whose role will be to ensure that the development process answers the requirements for this higher level of trust. These specialists might be responsible for writing the security policy of the system, as well as for creating the formal security policy model, and carrying out the formal specification and verification effort.

6.3.2.2 Subschemas

New subschemas Currently, SLCSE supports nine subschemas. Two of them are "administrative" (the Contract subschema and the Project Management subschema), i.e., they describe activities performed and documents used and created in the process of managing and administrating the project. Out of the remaining seven, six describe the activities and documents related to phases of the DoD-STD-2167A process, and one, the Environment subschema describes the features of the operational environment of a completed system.

For ITSDE, we see the need to add at least the following new subschemas:

- The Certification/Accreditation subschema.

  For this subschema, one can benefit from the effort that went into creation of the SCAP Workbench Tool, which is based on a similar idea of hypertext. It provides a good representation and organization of the activities mandated for certification/accreditation and the documentation that has to be produced, as well as the connections between the activities and documentation required by TCSEC.

- The Security Policy subschema.

  Various aspects of the system security policy (MAC, DAC, I&A, Audit) should be represented in this subschema at a level of detail sufficient for automatic generation of the security policy document. For trusted systems of higher levels of trust, the subschema will include entities corresponding to a formal policy model.

- The Specification and Verification subschema.

  For systems of higher levels of trust. This subschema will support such entities as Design_Specification, Code_Specification, Proof, etc. and connect them with appropriate links to the source code files.

- The Covert Channel Analysis subschema.

  This subschema will have entities corresponding to the methods of covert channel analysis supported by ITSDE such as shared resource matrix analysis.

- The Secure Operational Environment subschema.

  More analysis is needed to determine whether a new subschema is necessary or the existing Environment subschema can be augmented and modified to accommodate the mandated operational security analysis of the trusted system environment.

- Other schemas may be needed

Modifications of Existing Subschemas In the E-R representation of SLCSE subschemas, one can add entities and relationships between them, and also add or modify attributes of entities. In this section we give two examples of modifications to the E-R representation needed for ITSDE.

We will use the Software Requirements subschema as the first example. The current SLCSE Software Requirements subschema from [Str90] is shown in Figure 6.1. The entity CSCI_Capability describes the required functionality of the Computer Software Configuration Item (CSCI). Its attributes

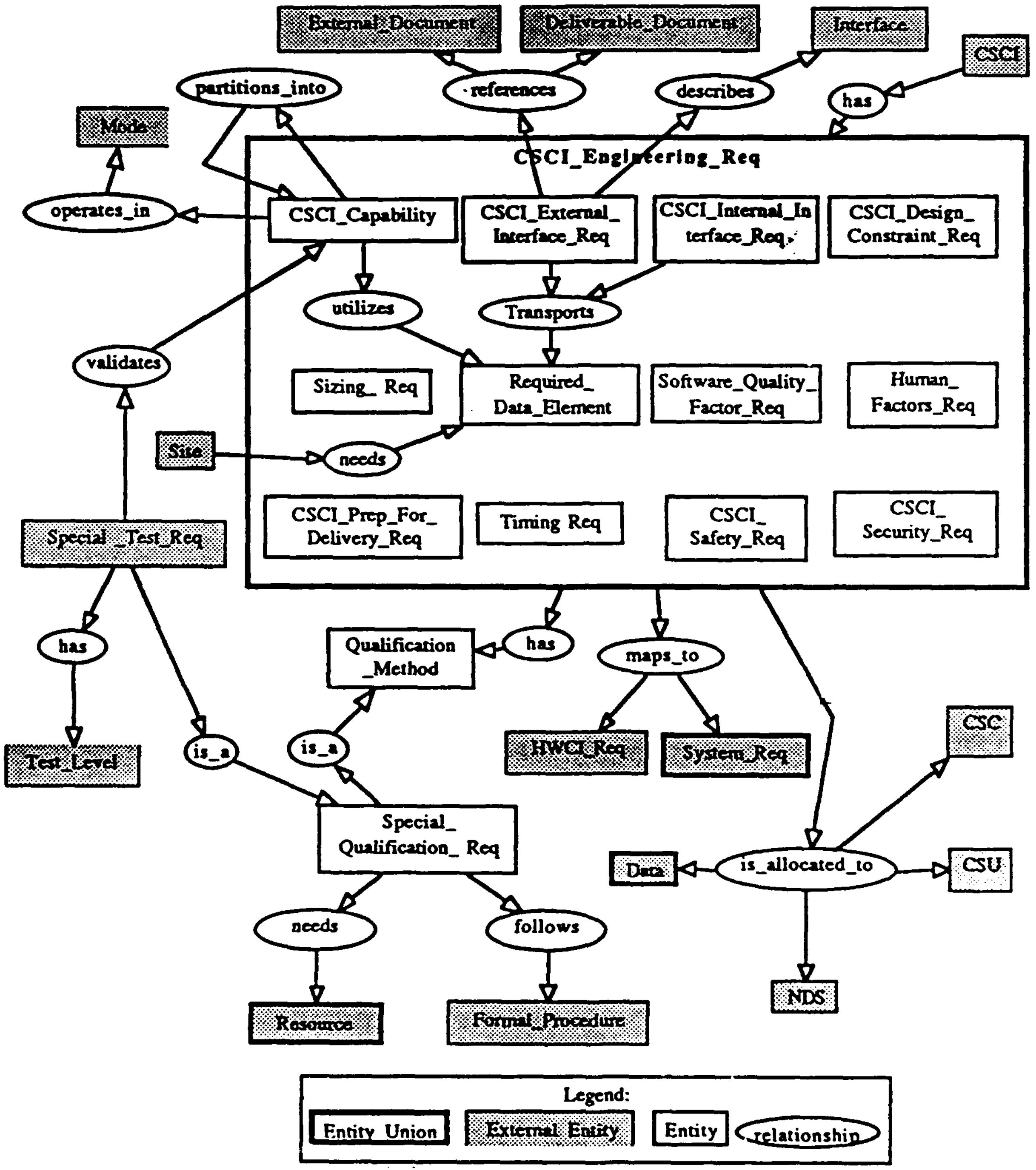

Figure 6.1: Software Requirements Subschema

are Purpose, Performance, Control_Flow_Diagram, and Data_Flow_Diagram [Str90]. The entity CSCI_Security_Req has only one attribute, Req_Statement, and is supposed to specify " security characteristics of the CSCI design with respect to potential compromise of sensitive data" [Str90].

For ITSDE, the following modifications of these attributes might be suggested. In trusted systems, the functional requirements of an trusted CSCI that corresponds to the system's TCB (or a part of the TCB) that implements a particular security service are "by definition" its security requirements. For such components, the CSCI_Security_Req entity becomes the value of the Purpose attribute of the CSCI_Capability entity.

In the E-SLCSE we might define a sub-object of a CSCI object, called a Trusted_CSCI, which will inherit all the general functional relationships of a CSCI, and also will have particular attributes specific for trusted components. For example, for trusted CSCIs of systems of higher levels of trust we might add a Formal_Security_Requirements attribute to the CSCI_Capability entity.

Our second example will be from the Design subschema. For systems of higher levels of trust, modifications to represent formal design specifications of trusted software components (CSCI, CSC, CSU) should be made to the Design subschema. Below we suggest tentative modifications as our second example.

Specifically, an external entity Formal_Top_Level_Specification (FTLS) (from the new Specification and Verification subschema) should be connected with all types of design components, CSCI, CSC, and CSU. Further analysis needs to be done to determine the best way to represent the Code-Model Correspondence requirement of the TCSEC. The activity of establishing Code-Model Correspondence is usually considered a part of the System Testing phase [Bod88]. One simple way is just to add a new Code-Model Correspondence entity in the Test subschema whose value is the correspondence argument and connect it by the appropriate relationships with the design components' entities (CSCI, CSC, CSU) in the design subschema (relationship "is_in") and with the FTLS entity in the Specification and Verification subschema (relationship "with").

For future "beyond A1" systems, where the Code-Model Correspondence is given as a formal argument (proof), the Code-Model Correspondence entity should be also connected with the Formal_Proof entity from the Specification and Verification subschema.

6.3.2.3 Integrating the Specification/Verification Tools into SLCSE

As we pointed out above, the attractive feature of implementing ITSDE on the SLCSE platform is that there should be a minimal amount of new software written for the integration of the specification/verification tools into SLCSE. In particular, the user should be able to invoke these tools via the ITSDE/SLCSE interface. In addition, the results of using these tools, when saved, should be written into the appropriate entities of the Specification/Verification subschema.

6.3.2.4 Documentation Generation

In Chapter 2 above we have pointed out that the software documentation DIDs will need to be tailored according to the process development variants supported by ITSDE threads, and we listed some sources of information for such tailoring.

Once the collection of documents supported by ITSDE and their thread-specific variants are established, the specifications for these documents in SLCSE should be produced. The specification of the documents in SLCSE will, in addition to fulfilling its own goal, serve a very important design function of "debugging" the ITSDE subschemas. In document generation specifications, each section of each document should have a pointer to an entity or an entity's attribute where the text corresponding to this section resides. The meticulous process of correlating entities and their attributes with sections of documentation will allow ITSDE developers to discover omissions, overlaps, a d other inadequacies of the newly defined or modified subschemas. Although tedious and time consuming, this part of develnping ITSDE is very straightforward.

Appendix A

THETA Mail Handler – Customer Goals

A.1 Introduction

This appendix contains a description of the customers' initial position about what they would like a mail facility on THETA to support. This appendix was prepared from the July 1, 1991 meeting at Rome Laboratory. (Rome Laboratory personnel agreed to play the role of the customer.)

A.2 Functionality

The functionality of the THETA mail facility can be divided into the following three areas:

- Sending a message
- Receiving a message
- Acknowledgements and notifications

A.2.1 Sending a Message

- Mail can be sent to either individuals or to a group of users.

- There should be a facility to aid in constructing replies to messages. Options for constructing replies to mail messages are:
  - Same as sending a message, but without having to fill in the destination.
  - Append reply to original message before sending it out.
  - Start up an editor with the original message and send out the edited message.
- Forwarding mail to other users should be convenient. An option to first use an editor would also be desirable.

A.2.2 Receiving a message

- Arriving mail can be pre-sorted by "project" or by arrival time. More sophisticated selections of mail, as in a database query, would be desirable.
- Mail can be directly printed without having to first save it or without having to have read it.
- Mail can be stored by users in different ways (for example in different files), but some automatic storage of mail should be provided (such as a system file, e.g., mbox).

A.2.3 Acknowledgements and Notifications

Users will be notified when new mail has arrived. Notifications may be in the form of just a beep, but it would be more desirable to have some text in some window. Another alternative is to display some graphical icon indicating that new mail has arrived.

When mail has correctly arrived, and possibly when mail has been read, the system will automatically send an acknowledgement to the sender.

A.3 User Interface

In general, an easy to use system is desired.

Particular Requested Features:

- When a mail item has been deleted, the remaining messages to be read will be renumbered starting with 1 and continuing consecutively.
- It should be possible to request that a mail message be printed directly from the user interface of the mail handler.
- Acknowledgements and notifications should arrive without cluttering the active window.
- The mail facility should be adaptable by having some kind of configuration file. One of the options should be where mail can be saved.
- Multiple users on the same system (but with different terminals) may be permitted.
- Some control over the color of the background and the type font, to indicate the classification level of the items, would be desirable.

A.4 Security

- MAC
  - The mail facility should not allow information at a given security level to be leaked either directly or indirectly to a user at a lower level. Even small channels from Notifications and Acknowledgements will not be permitted.
  - It would be desirable to minimize the amount of new MLS code.
  - Checking user clearances will be transparent.
- DAC

  The THETA file system DAC will be used for DAC access rights on saved mail.
- Privacy Enhancements

  Encryption will be a user option. Three options will be supported:

  - Request the THETA system to encrypt mail over the network.
  - Store unread mail in encrypted form.
  - Store read mail in encrypted form (with user keys).

- Constraints on Users

  Both single level and multilevel users may use the system.

  Mail activity in a given window will be single level (at the level of that window). Multilevel users will be responsible for not putting higher level information in a lower level window. Multilevel users will not be trusted to dynamically change the level of the window while in the mail handler.

- Assured Service

  Priority mail that blocks other services until the mail is read would be desirable. However, it should not be at the expense of MAC security leaks.

Appendix B

THETA Mail Handler – Preliminary Selection of Development Method

B.1 Introduction

This appendix is not complete but gives some idea of what is needed.

The Mail Handler will be added to the THETA Operating System and should conform to the THETA requirements for adding new managers.

For functional and security reasons it is natural to split the system into three parts (probably CSCI's). One component is the the User Interface that will display and input requests for sending and receiving mail. Another component is the Post Office for managing the mail that is to be delivered. The third component is the Notifier that will inform the user that there is mail to be read.

B.2 User Interface

Because of security considerations and customer requests, the User Interface should be single level.

B.3 Post Office

B.3.1 Multi-Single Level Solution at the COS Level

It is possible to build a mail handler as a COS extension on some node of the THETA system. This option is eliminated because it will not be a real extension to the THETA system (which is a customer goal). In particular, it will not be sufficiently portable to other kinds of COS nodes on the THETA system. However, if this option should be chosen at a later time, the design and security analysis should be based on the specific COS and security application criteria.

B.3.2 Multi-Single Level Solution at the THETA Level

Since the mail traffic is naturally separated by the level of the mail, one would suppose that a multiple single level handling of mail would be the desired solution. Indeed this would minimize the amount of trusted code. However, it is not such a good design with respect to the capabilities of the underlying operating system, i.e. THETA. This is principally because the number of processes would have to be much larger with a multiple single level approach.

B.3.3 Solution Using Existing THETA Autogeneration of Managers

There is one method described in [Cor92], that is the standard way of extending THETA. (This method is described in Chapter 5.) It does involve adding trusted code, but this addition is sufficiently small to be analyzable. (Detailed THETA documentation describing the advantages and disadvantages of this approach is still being developed.)

B.3.4 Alternative Restrictive THETA Manager

It is possible to derive a different security thread that would be restrictive, but this is unnecessary and would be costly.

B.3.5 Proposed Choice

Because it was desired that the design be COS independent, that the costly alternative of deriving a new security thread was not appropriate, and that the multi-single level solution would add too many additional single level processes, it was decided that the existing THETA Autogeneration of Managers method should be used.

Appendix C

Modeling for the THETA Mail Handler

In this appendixe, we describe a way of modeling the THETA Mail Handler for the requirements and top level design phases.

This appendix is not sufficiently developed to be used in a thread, but is sufficiently detailed to provide an idea of what could be done.

C.1 Formal Model for THETA Managers

In this model, we will assume that all of the data can be partitioned by label. That is, the data of the MLS process P can be represented as a product indexed by the level of the data.

Suppose that the MLS process P has variables xi.

```
/* the projection of the data at a particular security level */
   proj :: {data,level} -> data
/* Q -- a process which is supposed to be single level */
   Q :: {event,data} -> process
/* Guard  transformation -- assures  security independent of Q */
   Guard :: process -> process
/* update of process data */
   update ::  {data,event} -> data
/* the level of event e */
   lev :: event -> level
```

```
P(xi)= await e where validinput(e) then
        begin
           Guard( Q(e,proj(lev(e),xi)),lev(e));
           P(update(xi,e))
        end
```

such that
The invariant of the process includes:

- $update(xi, e)(l1) = xi(l1)$ for all $l1$ dominated by $level(e)$ (recall that type data is really an indexed product). This prevents even the existence of the message from effecting lower level data.

- for l dominating the level of e, $update(xi, e)(l) = update(proj(xi, l), e))(l)$. The updated version of the data at level l depends only on information at or lower than l.

Additionally

- Q does not handle inputs and eventually halts.

- the guarded process does not produce outputs at a level less than l. That is, $[\forall R, S : process][\forall l : level](Guard(R, l) \xrightarrow{\gamma} S$ and e in γ) $\Rightarrow$ $level(e)$ dominates l.

Such a process is restrictive (a special case of Romulus security condition generation (SCG) [Ros88]).

C.1.1 Simplification in the Case of No Read ups

In the case that the MLS processes do not need to make read ups, one can partition the data so that only the data at a particular level can be extracted.

So we can replace the proj(xi,level(e)) with data_at(level(e),xi).

C.2 Alternate Presentation

The above presentation was based on the current Romulus system. In the current version, processes are usually specified in a functional style, and in particular the data variables are viewed as parameters to the process. An alternative is to allow updates to the variables throughout the process definition.

If an assignment operation is added to the SL language, then we could combine the updating of the variables with the production of the outputs. This will permit the specification to more nearly follow an imperative style design (i.e., like a traditional programming language).

In this case we could define the *Guard* transformation to block any updates to lower level data. That is, instead of having a separate trusted *update* function, it would be incorporated into the one guard.

In the current version of Romulus this is a bit awkward to express. However, Romulus is currently being revised to handle problems like this one.

C.2.1 Summary of Constraints

There are two kinds of objects messages and parameters.

Rules for Messages:

- No Read Up. Is imposed on Q by the formalism. (The data passed to Q is at or below the level of the input.)
- No Write Down. Is explicitly imposed on Q by the Guard. All data are labeled at least at a level greater than or equal to the input.

Rules for parameters:

- No Read Up. Is imposed on Q by the formalism (The data passed to Q is at or below the level of the input) and
- No write down. Any assignment $xi(l) := ...$ is blocked by the Guard for any l such that *not dominates*$(l, level(e))$. (Any internal transition event to do an update will explicitly be designated at some l, and if that level is not allowed, the process will continue without doing the update.)

C.3 Design and Implementation Alternatives

In this section, we briefly discuss how to produce a design and implementation that will be restrictive. There is more than one possible method and we describe several methods in the next subsections.

The goal in the development of the next stages is to minimize the amount of work that the designers and implementors have to perform in order to achieve restrictiveness with high assurance. However, the method chosen will depend on what sort of automated tools are available.

C.3.1 Do Not Use a Guard

A traditional approach is to construct a process Q that doesn't need a guard. That is build Q such that $Guard(Q(e, proj(lev(e), xi)), lev(e)) = Q(e, proj(lev(e), xi))$.

C.3.1.1 Build and verify an FTLS

We can construct a detailed formal description of Q and use Romulus to check that outputs are labeled correctly and that the data parameters are also handled correctly. Implicit in this analysis is that the underlying system can assure correct delivery of messages and that other processes do not change the process's data, i.e., xi.

C.3.1.2 Code Level Verification

Alternatively, we can generate the constraint as in the FTLS verification approach, but defer the verification of the constraints to a verification of constraints on the code. One would still inspect the constraints at the design level, but one would not carry out any formal proof. Each design constraint on an output or data parameter would be converted into a corresponding code constraint. While it is possible, that a difficult problem could be pushed to the code level analysis, for this class of models it is unlikely. Also, we will still need to check the code for potential security violations.

C.3.2 Use a Guard

Another approach is to design and build a guard that filters or corrects the labels of outputs and protects the multilevel data. The guard must be securely informed of the level of the input e. Any output of Q can then be corrected or discarded based on that level. Similarly, updates and accesses to the data can be mediated by the guard. An analysis would still have to be made that the guard was correct and protected, but we would no longer have to verify anything about Q.

C.3.3 Method for the THETA Mail Handler

The THETA Mail Handler was chosen to be built as a weak-MLS manager using the autogeneration methods of THETA. It mostly uses the guard method. Events that are invocations to other managers are sent through the Kernal which checks the current operating level of the manager, for that invocation, to adjust the label. Events which are system calls are implicitly associated with the current operating level of the process. The guard protects the memory by the way data is accessed and updated in the autogeneration method. However, the guard does not completely protect the Q part of a manager from tampering or inappropriately accessing data at a different level from which it was invoked. This is handled by checking that the code was produced by the autogeneration method and that it only utilizes the allowable access methods.

Appendix D

System Specification

The following appendix is a first approximation of the System/Segment Specification for the THETA Mail Handler and does not incorporate all of the *suggestions* by TIS (as their report arrived too late).

SYSTEM SPECIFICATION
FOR THE
THETA Mail Handler

D.1 Scope

D.1.1 Identification

Number: xxx, THETA Mail Handler

D.1.2 System Overview

The purpose of the THETA Mail Handler is to securely extend the THETA distributed operating system with a facility to send and receive mail messages between users of the THETA system.

D.1.3 Document Overview

This document is a preliminary specification of the THETA mail handler system requirements.

D.2 Applicable Documents

D.2.1 Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

1. "Trusted Computer System Evaluation Criteria", Department of Defense,DoD-5200.28-STD, Dec. 1985.
2. "Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria", National Computer Security Center,NCSC-TG-021, version 1, April 1991.

D.2.2 Non-Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

1. "A Hookup Theorem for Multilevel Security",Daryl McCullough, IEEE transactions on Software Engineering, Jun 1990, volume 16, number 6, pages 563-568.
2. THETA System Requirements documentation for constructing New Managers (not currently available).

D.3 System Requirements

D.3.1 Definition

D.3.1.1 System Configuration

The THETA Mail Handler will have its functionality split between two pieces: a user interface and the mail manager. (It has not yet been determined whether the mail manager will be a manager in the sense of a THETA manager or whether it will be a THETA application that uses the THETA File manager to control its objects.)

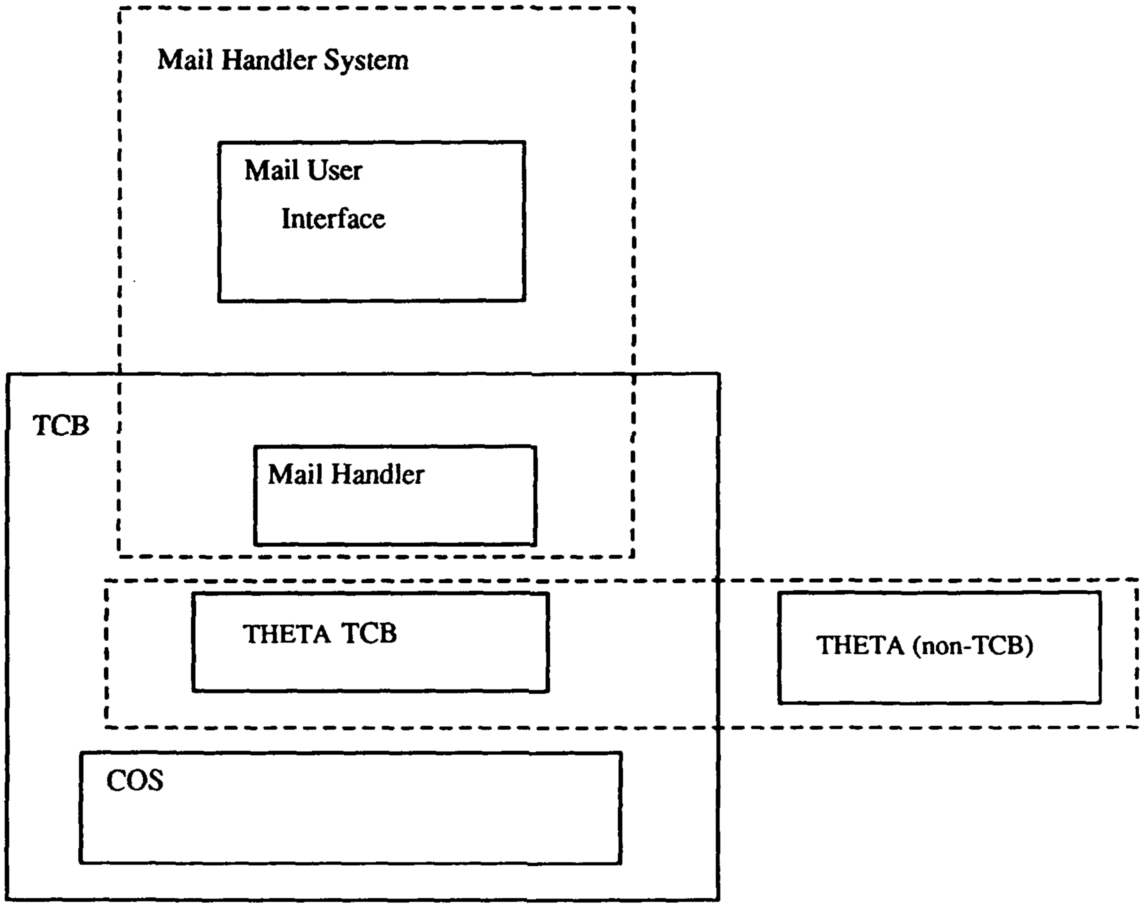

Figure D.1: System Configuration

D.3.1.2 Functionality

The purpose of the THETA Mail Handler is to facilitate the exchange of information between users of the THETA system, subject to the security requirements on the users and their messages. This facility is only for communication between and internal to THETA sites on a given THETA system.

D.3.1.2.1 Functionality For the purposes of exposition, the functionality of the THETA mail facility can be divided into the following three areas:

- Sending a message
- Receiving a message
- Acknowledgements and notifications

D.3.1.2.1.1 Sending a Message

- Mail will be sent to either individuals or to a group of users.
- There will be a facility to aid in constructing replies to messages. Possible options for constructing replies to mail messages may be
  - Same as sending a message, but without having to fill in the destination.
  - Append reply to original message before sending it out.
  - Start up an editor with the original message and send out the edited message.
- Forwarding mail to other users should be convenient. An option to first use an editor will be available.

D.3.1.2.1.2 Receiving a Message

- Arriving mail can be pre-sorted by "project" or by arrival time. More sophisticated selections of mail may also be supported.
- Mail can be directly printed without having to first save it or without having to have read it.
- Mail can be stored by users in particular files at the user discretion. In addition, some automatic storage of mail will be provided (such as a particular file, e.g., mbox).

D.3.1.2.1.3 Acknowledgements and Notifications Users will be notified when new mail has arrived. Notifications may be in the form of just a beep, but it will also include options for displaying notifications as text in some window and as some graphical icon indicating that new mail has arrived.

When mail has correctly arrived, and possibly when mail has been read, the system will automatically send an acknowledgement to the sender.

D.3.1.2.2 User Interface In general, the user interface should be easy to use.

Particular Requested Features:

- When a mail item has been deleted, the remaining messages to be read will be renumbered starting with 1 and continuing consecutively.
- It will be possible to request that a mail message be printed directly from the user interface of the mail handler.
- Acknowledgements and notifications will arrive without cluttering the active window.
- The mail facility will be adaptable by having some kind of configuration file. One of the options will be where mail can be saved.

D.3.2 Characteristics

D.3.2.1 Performance Characteristics

There is only one state and mode for the Mail Handler. However, the THETA system may be reconfigured without a Mail Handler. All the capabilities described above assume that the Mail Handler is active.

D.3.2.1.1 (State Name) Not Applicable

D.3.2.2 System Capability Relationships

Not Applicable

D.3.2.3 External Interface Requirements

D.3.2.3.1 THETA Mail Handler External Interface Description This system will be connected to the THETA system. Data passed from the trusted Mail Handler code to other parts of the THETA system must be appropriately labeled.

D.3.2.4 Physical Characteristics

Not Applicable

D.3.2.5 System Quality Factors

D.3.2.5.1 Reliability THETA supports added reliability through replication. It is transparent to the design of the Mail Handler.

D.3.2.5.2 Maintainability Standard THETA Software Maintenance practices should be used.

D.3.2.5.3 Availability No special availability conditions must be met. Continuous availability of the THETA system and the Mail Handler is a desired goal.

D.3.2.5.3.1 Additional Quality Factors None

D.3.2.6 Transportability

Not Applicable

D.3.2.7 Flexibility and Expansion

The THETA Mail Handler should be added to the THETA system consistent with the THETA guidelines for adding new managers.

The user interface should be designed so that it can be upgraded to handle other COS and THETA window and mouse support.

D.3.2.8 Portability

Not Applicable

D.3.3 Design and Construction

Not Applicable

D.3.3.1 Materials

Not Applicable

D.3.3.1.1 Toxic Products and Formulations Not Applicable

D.3.3.2 Electromagnetic Radiation

Not Applicable

D.3.3.3 Nameplates and Product Marking

Not Applicable

D.3.3.4 Workmanship

Not Applicable

D.3.3.5 Interchangeability

Not Applicable

D.3.3.6 Safety

Not Applicable

D.3.3.7 Human Engineering

The user interface design should attempt to minimize the chances of a user inadvertently entering high level information while using the mail handler at a lower level. (General guidelines are not yet available.)

D.3.3.8 Nuclear Control

Not Applicable

D.3.3.9 System Security

The system should be B3 (TCSEC), but the interpretation of these requirements may be adjusted to reflect the contracting agency's objectives. (The certification and accreditation procedures will be based on the Air Force procedure xxxyyyzzz.)

- MAC
  - The mail facility should not allow information at a given security level to be leaked either directly or indirectly to a user at a lower level. Even small channels from Notifications and Acknowledgements will not be permitted.
  - The amount of new MLS code will be minimized, except that efficiency considerations can be used to override this objective.
  - Checking user clearances will be transparent.
- DAC

  The THETA file system DAC will be used for DAC access rights on saved mail.
- Privacy Enhancements

  Encryption will be a user option. Three oprtions that will be supported are:
  - Request the THETA system to encrypt mail over the network.
  - Store unread mail in encrypted form.
  - Store read mail in encrypted form (with user keys).
- Constraints on Users

  Both single level and multilevel users may use the system.

  Mail activity in a given window will be single level (at the level of that window). Multilevel users will be responsible for not putting higher level information in a lower level window. Multilevel users will not be trusted to dynamically change the level of the window while in the mail handler.

- Assured Service

  Priority mail that blocks other services until the mail is read may be partially supported. However, it will not be at the expense of MAC security leaks. This feature should either be implemented or a report should be prepared indicating why this feature was not supported.

D.3.3.10 Government Furnished Property Usage

The Mail Handler is intended to be used on a THETA System (contact Rome Laboratory).

D.3.3.11 Computer Resource Reserve Capacity

No special requirements.

D.3.4 Documentation

Security documentation will be developed as required by the ITSDE version of 2167A for secure systems.

D.3.5 Logistics

No significant impact on existing maintenance, facilities, or equipment.

D.3.6 Personnel and Training

Appropriate installation and auditing procedures will be followed.

D.3.6.1 Personnel

No new personnel are needed.

D.3.6.2 Training

Not applicable

D.3.7 Characteristics of Subordinate Elements

The THETA Mail Handler contains no Segments.

D.3.8 Precedence

Nondisclosure should take precedence over functionality requirements provided the resulting system will be usable.

D.3.9 Qualification

D.3.9.0.1 Security Analysis

1. Formal Model and Verification
2. Covert Channel Analysis
3. Penetration Test

D.3.9.0.2 Functionality Analysis

1. Testing of User Interface, Mail Manager, and integrated system

D.3.10 Standard Sample

Not Applicable

D.3.11 Reproduction Sample, Periodic Production Sample, Pilot, or Pilot Lot

Not applicable

D.4 Quality Assurance Provisions

Assurance provisions for security will be achieved through the accreditation process (and the use of an approved THETA development thread).

D.4.1 Responsibility for inspection

The Certification will be handled by XXX.

D.4.2 Special Tests and Examinations

Not applicable

D.4.3 Requirements Cross Reference

Not included in this sample SSS.

D.5 Preparation for Delivery

No special preparations are necessary.

D.6 Notes

Abbreviations:

- B3 - The B3 level of security (in TCSEC)
- MLS - Multilevel Secure
- TCB - Trusted Computing Base
- TCSEC - Trusted Computer System Evaluation Criteria
- TDI - Trusted DataBase Interpretation
- THETA - Trusted Heterogeneous Architecture (a secure distributed operating system)

D.6.1 Intended Use

The intended use of the Mail Handler is to exchange mail between users of the THETA system.

D.6.1.1 Missions

Not applicable

D.6.1.2 Threat

Security threats are not included in this sample SSS.

Bibliography

[BBC90] M Branstad, W. C. Barker, and P. Cochrane. The role of trust in protecting mail. In *IEEE Computer Society Symposium on Research in Security and Privacy*, May 1990.

[Ben89] T. Benzel. Developing trusted systems using DoD-STD-2167A. In *Fifth Annual Computer Security Applications Conference*, December 1989.

[Bod88] Deborah J. Bodeau. Guidance for reviewers of security-relevant design specification and verification documentation. Technical Report MTR-10478, Mitre, Bedford, Mass, Sept 1988.

[Boe88] Barry W. Boehm. A spiral model of software development and enhancement. In *Computer*, pages 61–72. IEEE, May 1988.

[BR93] Terry Benzel and Doug Rothnie. Integrated Trusted System Development Environment — Process, Final Report. for contract F30602-89-C-0049, RL-TR-93-xxx, in preparation, 1993.

[Cor92] ORA Corporation. Manager developer's tutorial. Technical Report Contract F300602-88-0146, CDRL A024, For Rome Laboratory, April 1992.

[Dep88] Department of Defense. *Military Standard — Defense System Software Development*, February 1988. DoD-STD-2167A.

[DoD85] Department of Defense. *Trusted Computer System Evaluation Criteria*, December 1985. DoD-5200.28-STD.

[GM82] Joseph A. Goguen and José Meseguer. Security policy and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20, Oakland, CA, April 1982. IEEE.

[Gua89] David Guaspari. Penelope, an Ada verification system. In *Proceedings of Tri-Ada '89*, pages 216–224, Pittsburgh, PA, October 1989.

[McC90] Daryl McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, June 1990.

[NCS87] National Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, July 1987. NCSC-TG-005, version 1.

[NCS91] National Computer Security Center. *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, April 1991. NCSC-TG-021, version 1.

[NIS92] NIST. *Minimum Security Functionality Requirements for Multi-User Operating Systems*, January 1992. Draft Version.

[ORA90] ORA Corporation. Romulus: A computer security properties modeling environment, final report - volume 1: Overview. Technical report, ORA Corporation, June 1990.

[RL992] ORA Corporation. *Experimental Secure Distributed Operating System Development — THETA Phase II*, June 1992.

[Ros88] David Rosenthal. An approach to increasing the automation of the verification of security. In *Proceedings of Computer Security Foundations Workshop*, pages 90–97, Franconia, NH, June 1988. The MITRE Corporation, M88-37.

[Str90] Tom Strelich. Software life cycle support environment. Technical Report RADC-TR-89-385, General Research Corporation, February 1990.

[Sut92] Ian Sutherland. Shared-state restrictiveness. Romulus Project, 1992.

[TIS89] TRW, Computation Logic Inc., and Trusted Information Systems. Process model for high performance trusted systems in Ada. Technical Report ARPA 6414 Contract MDA 972-89-C0029, TRW, August 1989.

*U.S. GOVERNMENT PRINTING OFFICE: 1993-710-093-60276