Technical Report

CMU/SEI-95-TR-015
ESC-TR-95-015
October 1995

The Unified Information Security (INFOSEC) Architecture

Fred Maymir-Ducharme

P.C. Clements

Kurt Wallnau

Robert W. Krut, Jr.

Technical Report
CMU/SEI-95-TR-015
ESC-TR-95-015
October 1995

The Unified Information Security (INFOSEC) Architecture

Fred Maymir-Ducharme
P.C. Clements
Kurt Wallnau
Robert W. Krut, Jr.

Application of Software Models

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

List of Tables

The Unified Information Security (INFOSEC) Architecture (UIA) Gadfly Project

Abstract: This report captures the development, lessons learned, and future recommendations from a collaborative research and development activity between the Air Force sponsored Comprehensive Approach to Reusable Defense Software (CARDS) Program, the Department of Defense (DoD), and the Software Engineering Institute (SEI). This activity explored innovative but practical techniques for formalizing and applying (i.e., reusing) models of information security (INFOSEC) concepts to the development of information systems.

1 Introduction

Information Security (INFOSEC) technical concepts and best practices have emerged through many years of developing secure information systems, but the management, transmission, and application of this knowledge is informal. To capture and organize this knowledge in the form of models, and support the systematic application (i.e., reuse) of this knowledge to the design of secure information systems, the Department of Defense (DoD) developed a structured body of knowledge, called the Unified INFOSEC Architecture (UIA), and sought to apply this domain expertise in the design, redesign, evaluation, and maintenance of information systems.

The DoD wanted to demonstrate the practical application of the UIA. Since INFOSEC principles are most effective when they are designed into an application rather than retrofitted, it made sense to apply the UIA to high-level application designs (i.e., *architectures*). Therefore, a practical application of the UIA would demonstrate a mapping between UIA models and application-specific information system architectures (such as command centers).[1] Such a mapping might consist of online assistance in the application of UIA principles during system design, or, alternatively, might consist of tools to enable *post hoc* evaluation of system designs from a UIA perspective.

To demonstrate the practical application of the UIA, a collaborative research and development activity was established between the Air Force sponsored Comprehensive Approach to Reusable Defense Software (CARDS) Program, the DoD, and the Software Engineering Institute (SEI). Specifically, this collaboration was established to explore innovative but practical techniques for the formalization and application of the UIA concepts to software architectures.

1. "Architecture" is used to mean different things in UIA and application-specific information system architectures. In the former, the term refers to a body of knowledge. In the latter, it refers to a high-level design for a family of systems.

The remainder of this introductory section describes the background of the collaboration (Section 1.1), the collaborative objectives (Section 1.2), the purpose of this report (Section 1.3), and the structure of this report (Section 1.4).

1.1 Background of Collaboration

The DoD INFOSEC System Engineering Process addresses principles of access control, integrity, and assurance of security. The DoD developed UIA are conceptual models of INFOSEC principles and processes for applying these principles to the design of secure information systems. The DoD desired to define more clearly the structure and content of these INFOSEC models, and to define technically sound and repeatable methods for applying these models to select and analyze architectures for the design of secure information systems. CARDS and the SEI view this problem primarily from a reuse perspective; that is, they understand the problem to be one of

- formally representing expertise in the INFOSEC domain in models
- using these models during the specification and design (or evaluation of designs) of information systems from a security point of view
- providing an enactable process for reusing the models (i.e., tool-support)

This kind of high-level reuse is consistent with the model-based approach espoused by the SEI, CARDS, and others. The challenges addressed by this research effort are

1. To use domain analysis methods to model the process followed by security analysts in developing or evaluating the security attributes of a system.
2. To understand the relationship between the UIA and software architectures for secure systems. In addition to modeling INFOSEC principles, this required formally modeling what was meant by software architecture (an *abstract* model), how specific designs instantiate these models (a *concrete* model), and how INFOSEC knowledge can be mapped to both abstract and concrete models.
3. To develop tools to model the relationship and support decision making during specification and design.

The DoD, CARDS, and the SEI agreed to work collaboratively to develop a proof of concept *automated reuse assistant* to map the UIA to abstract and concrete models of software architecture; i.e., to demonstrate the application of UIA principles to the creation and/or evaluation of security concerns within domain-specific software architectures. This UIA reuse assistant is referred to as the *UIA Gadfly*.

1.2 Collaboration Objectives

To meet the overall objectives of this collaboration, the DoD, CARDS, and the SEI agreed to apply and extend model-based technologies developed by the SEI and the CARDS program. These technologies will demonstrate how software models and tools can aid system security architects in the process of generating candidate architectures that provide reusable INFOSEC solutions to information systems.

In particular, INFOSEC models and software architectures can be used to help system security architects analyze threats to a system or component, and produce recommendations on

- missing threat information they need to be concerned about (but did not specify)
- lists of primary & supporting security “services” to counter threats
- discovering the complex interrelationships between various threats, threats and services, and when they can be combined. The ultimate goal is to form lattices of primary and supporting security services to counter all threats.

The specific objectives of this collaboration were to

1. Investigate and illustrate the utility of an INFOSEC domain model to support INFOSEC system development and analysis.
2. Apply *Feature-Oriented Domain Analysis* (FODA)[1] [Kang 90] to formalize the process of security analysis using UIA models.
3. Develop the UIA automated reuse assistant from existing CARDS model-based technology.

To accomplish these objectives, a collaboration team was composed of representatives from the DoD, the SEI, and the CARDS program. Significant responsibilities in providing domain expertise and knowledge encoding would fall to the DoD participants; additionally, DoD would provide critical input to scenario definition and overall system concept definition. The SEI would provide guidance on the use of FODA, and would provide critical review of the overall technology effort, especially as it relates to the products of the domain analysis effort and the application of these products to the development of automated reuse tools. CARDS would provide expertise and guidance on knowledge encoding principles, perform detailed system design, and lead in the proof of concept development efforts.

Team members worked from their own geographically dispersed locations, but met for regularly scheduled Technical Interchange Meetings (TIMs),[2] and working meetings to determine initial scenarios and share status on the proof of concept demonstration.

1. Feature-Oriented Domain Analysis is an SEI-developed domain analysis technique.

2. See Appendix A for milestone chart.

1.3 Purpose of the Report

The purpose of this report is to provide an overall summary of the UIA Gadfly effort. Specifically, this report

- presents the basis of the collaboration
- presents the key technical concepts
- describes the approach taken
- describes the proof of concept demonstration produced
- provides lessons learned
- provides recommendations for future work

1.4 Structure of the Report

This report is structured as follows: Section 2 describes the approach taken for the development of the UIA automated reuse assistant. This section includes a description of a previously developed automated reuse assistant, a description of how these previous design concepts can be applied to the UIA domain, and a description of the method employed to define and capture the security analysis process. Section 3 provides an overview of the UIA Gadfly demonstration. Section 4 briefly describes the lessons learned and provides recommendations for future directions and extensions of the Gadfly approach. Finally, Section 5 provides the report conclusions.

2 The UIA Automated Reuse Assistant

The CARDS program has explored various ways to exploit the knowledge captured and the assets stored in model-based repositories such as the CARDS' Generic Command Center Library (GCCL). The CARDS library uses the Reuse Library Framework (RLF) [STARS 92, STARS 93a, STARS 93b, Wallnau 88] as the mechanism to encode information about building software systems for command centers.[1] In the Command Center Library, information about building command centers is represented by RLF as a structured inheritance network. A CARDS prototype system composition tool was developed to assist a designer in composing command center systems from components stored in the library. This System Composition Tool (or composer), written in CLIPS [Giarratano 93, NASA 93a, NASA 93b, NASA 93c], is a forward chaining, rule based system, that 'walks' the network, assisting the user in making appropriate selections while composing a new system.[2] This automated, RLF-based, reuse assistant, for system composition, became the primary technology for exploiting RLF-encoded domain models.

2.1 The UIA Gadfly

Recently, the RLF-based reuse assistant has been extended to support the integration of, and reasoning from, separately modeled and maintained RLF knowledge bases.[3] This effort integrated a black-box test generation knowledge base with an Ada abstract data type knowledge base to assist Ada programmers in developing black-box testing plans for Ada packages. This assistant is known as the "Black Box Gadfly" (Figure 2-1).[4]

For the UIA Gadfly, the RLF-based assistant was initially created by the integration of, and reasoning from, the INFOSEC knowledge base and the CARDS Command Center knowledge base. However, these knowledge bases lacked information crucial to the assistant. They lacked information about the environmental context of the application (i.e., they lacked information about understanding the security analysis process). In order to develop the environmental context knowledge base, FODA was used to help develop and expand several scenarios that highlighted relevant information. This enabled the team to develop a good model of the features of the security analysis domain that were relevant to the goal of the UIA Gadfly project.

1. The DoD used the RLF to encode a model of INFOSEC.

2. Appendix B provides a description of the CARDS System Composer.

3. A knowledge base is a set of models that represent the domain knowledge.

4. Appendix C provides a description of the Black Box Gadfly.

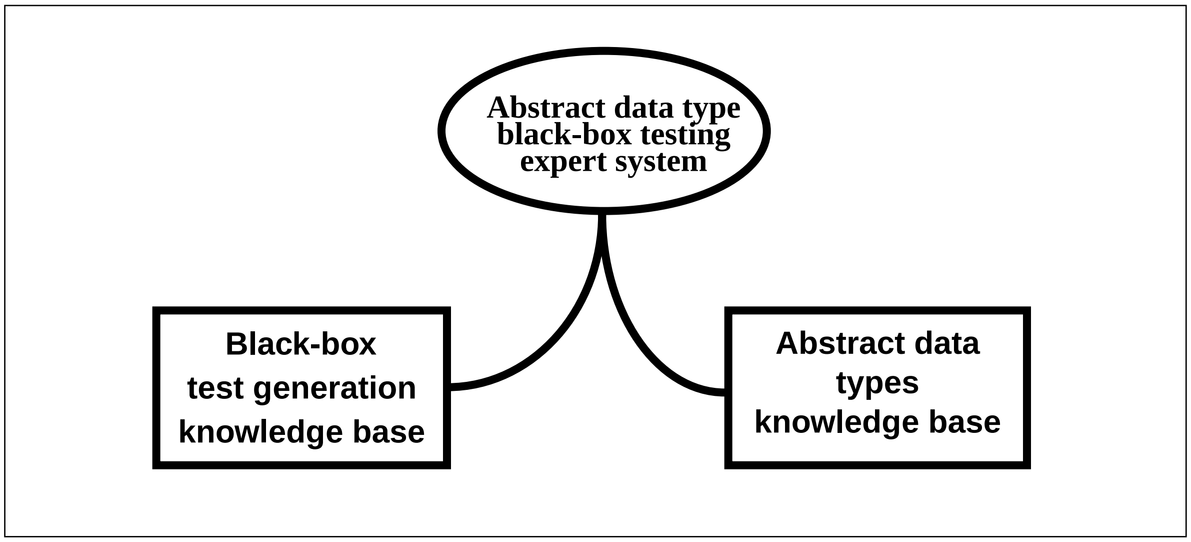

Abstract data type black-box testing expert system
Black-box test generation knowledge base
Abstract data types knowledge base

Figure 2-1: The Black-Box Gadfly

Once the environmental context knowledge base was created to elicit concept of operations and environment, it was linked to the existing Command Center and INFOSEC knowledge bases to form the UIA Gadfly (Figure 2-2).

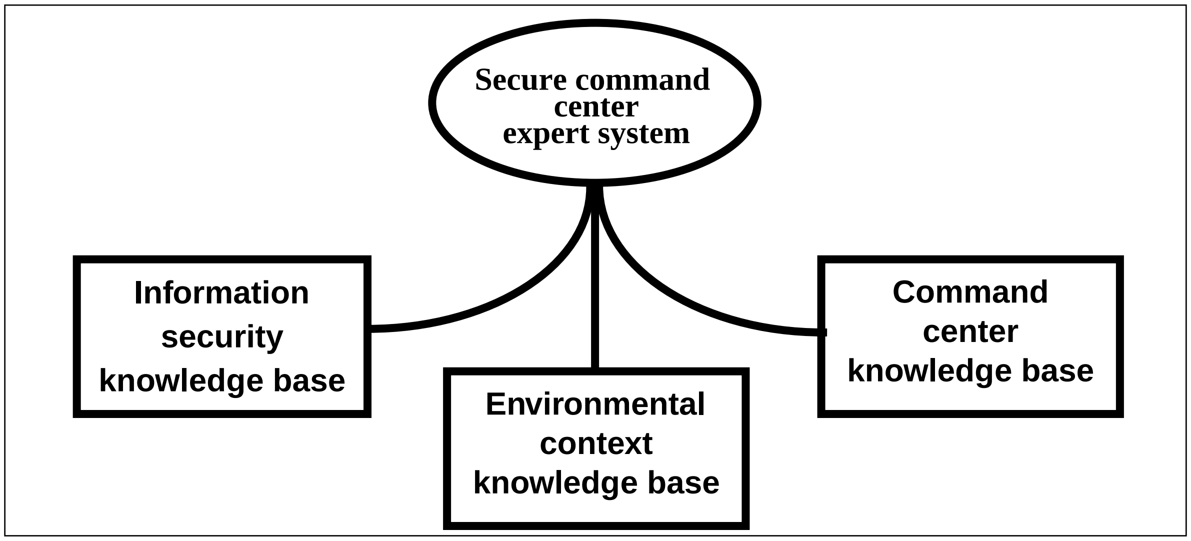

Secure command center expert system
Information security knowledge base
Environmental context knowledge base
Command center knowledge base

Figure 2-2: The UIA Gadfly

The following subsection provides the rationale for, and development of, the environmental context knowledge base.

2.2 Development of the Environmental Context Knowledge Base

The environmental context knowledge base was developed to provide an understanding of the security analysis process. The initial information was captured as part of the “FODA Models for INFOSEC System Design” workshop. This workshop was designed to establish a framework of information that captures the INFOSEC system design and security analysis activities in a domain model. The environmental context knowledge base was developed from the initial workshop information and follow-up elicitation of security analysis domain experts. The results of this effort provided a knowledge base used to elicit users about the context and environment of an application.

2.2.1 Understanding the Security Analysis Process

Security analysis supports the development of applications that are secure from [Gulachenski 94]

- an individual or entity gaining access to information they are not authorized to receive
- any circumstance or event that may result in an authorized individual or entity receiving false information that is believed to be true
- any circumstance or event that interrupts or prevents the correct operation of system services and functions
- any circumstance or event that results in the control of system services or functions by an unauthorized individual or entity

This analysis must consider a system’s security in terms of either what the system must be protected against or what type of protection the system is expected to provide. The security analyst must understand the system’s security objectives, security threats, and particular types of protection the system uses to counter the threats. By describing a system’s security needs from the INFOSEC perspective, the analyst gains an understanding of the high-level threats that need to be considered when assessing a system’s security needs, as well as the security services that can be used to satisfy these needs.

The UIA captures INFOSEC principles and processes for applying these principles to the design of secure information systems. The UIA models do not capture the process followed by a security analyst in developing or evaluating the security attributes of a system; however, this process must be captured to demonstrate how software models can aid system security analysts in the process of generating reusable INFOSEC solutions to information systems.

Security analysts follow a set of procedures to access security related attributes of a system. Developing a reusable process for these analysts would require

- understanding inputs to process and products of security analysis
- defining the operations followed by the analyst
- modeling the issues and concerns
- understanding when and how the process may be modified (e.g., Has the system been built and is security to be retrofitted?)
- modeling the process

A domain analysis of the process can yield these results.

The domain analysis methodology employed to understand the security analysis process was FODA.[1] The artifact of discussion is a *scenario*. FODA was used to help develop and expand scenarios under which a subscriber solicits the help of the organization to design or field a system that satisfies its security requirements; or solicits the help of the organization to assess the security fidelity of an already designed or fielded information system.

2.2.2 Applying FODA to Capture the INFOSEC System Design and Security Analysis Activities

Appendix E provides examples of the use of FODA to understand the security analysis process for the activity of the DoD branch that provides security analysis of information systems under the INFOSEC security model. These examples were a result of the context analysis phase of FODA, which defines the extent (or bounds) of a domain for analysis, and the domain analysis phase of FODA, which provides a description of the problem space in the domain. The context analysis phase captures such information as the inputs and outputs of the process of security analysis. Inputs include information about the system to be analyzed (mission constraints, threats), the intent of the effort (problem definition), and ongoing infrastructure constraints (policy), while the outputs include the security recommendations and relative risks. The domain analysis phase captures information such as the entities, capabilities (features), and operations involved in security analysis (e.g., what are threats, services, and mechanisms; what are the capabilities of a threat or vulnerability analysis; what are the types of threats to a system; what are the functions involved in security analysis).

1. Appendix D provides a brief description of FODA.

The environmental context knowledge base consisted mostly of rule-based information designed to capture the security analysis process and link this information to INFOSEC system design. Of particular interest to this effort was the *feature model* produced by FODA and associated knowledge about the interpretation and application of this feature model to the design of applications. The features model

- Describes various kinds of semantic relationships among features (e.g., mutual exclusion, aggregation, specialization).
- Captures rationale for the selection of features under different circumstances.

Features describe the context of domain applications, the needed operations and their attributes, and representation variations; thus, features and the features model represent the link to variations and decisions (i.e., rules) within the environmental context knowledge base and to the UIA model. The UIA topic areas, such as those provided in Appendix F, form the basis for the rules that tie the UIA model to the environmental context knowledge base.

The environmental context knowledge base and its links to the Command Center and INFOSEC knowledge bases were developed from

- the FODA information
- additional elicited security analysis domain information

3 The UIA Gadfly Demonstration

The following sections describe the UIA Gadfly demonstration and lessons learned.

The UIA Gadfly demonstration[1] shows how a software component for a secure system could be configured, while taking into account information relevant to the security of the system. The specific software component used for the demonstration is a message-processing unit in a command and control system, which operates in an environment of communication with outside, mobile personnel such as those used by law enforcement personnel in a drug raid.

Appendix G provides detailed information about the major parts of the demonstration and the information provided by and reported to the user. The following provides a brief look at a session with the UIA Gadfly. In general, a session would proceed as follows:

1. Questions are asked to determine with which of the possible threat consequences (e.g., “disclosure”) and specific threat actions (e.g., “disclosure via interception via wiretapping”) the application engineer is concerned.
2. Questions are asked about the type of command center the application engineer wishes to build; components are selected from the command center domain knowledge base, and the architecture for a command center system is assembled.
3. Security risks are associated with each of the components in the command center domain library. For example, if the command center is physically distributed along communication lines, then it is inherently vulnerable to disclosure of various sorts. If the application engineer is concerned about disclosure of information, then the Gadfly *infers* ways in which the chosen architecture is inherently vulnerable to disclosure threat actions, whether or not the engineer lists them specifically.
4. The lists of threats (stated by the application engineer and inferred by the Gadfly) is compiled.
5. A set of appropriate security measures is suggested to counter the applicable threats.
6. The application engineer can review his or her choices, assemble a different command center architecture, and investigate how alternative choices affect the information security of the planned application.

1. All work and the demonstration was done on Sun SPARC stations running SunOS and Unix. The CLIPS version used was CLIPS 6.0 for the Unix platform.

4 Lessons Learned

The following section provides the lessons learned from the UIA Gadfly demonstration. Included in this section are recommendations for both extending the UIA Gadfly proof of concept and the Gadfly approach.

The UIA Gadfly demonstration was very informative; the most important lessons learned are summarized below, followed by recommendations for future Gadfly efforts.

The most valuable lesson learned from the demonstration is that it is feasible to develop an assistant such as that proposed. The assistant enforces the process defined during the initial domain analysis effort. It draws on both a product/system model (GCCL) and an INFOSEC model to provide information about the security of a proposed software system. This demonstration shows how the two models can be linked together, with knowledge drawn from both, to produce the final system. Thus the proof of concept was highly successful.

On a long-term basis it would be desirable to provide a system that would

1. allow the user could look at a threat
2. choose a security mechanism
3. re-prioritize remaining threats within the context of that mechanism

This would involve both lowering the priority of some threats because the mechanism deals partially or wholly with the threat, and increasing the priority of others (and even adding new threats) because of side effects of the security mechanism. In essence, a desirable assistant feature would “configure” a secure software system, and interactively modify the remaining threats as security features are added.

Another long-term goal is to have a system that allows the user to see the impact on security of choosing specific domain components, and to modify the choices as appropriate. If requested, the UIA Gadfly system will now report on inferred threats before a choice is made, so in a single step we have reached this goal. To implement it for the entire system, however, requires the ability to change earlier choices. This would require adding an *UNDO* capability to the system (or the mutual exclusion rules from the domain model features).

4.1 UIA Gadfly Recommendations

A number of additional areas were identified as valuable extensions that are beyond the scope of this proof of concept, but would be appropriate for additional work. This subsection addresses these proof of concept extensions.

The major recommendation is to extend the proof of concept to a prototype and eventually to a full system. Recommendations for additions to the proof of concept include:

- re-casting environment rules into structured knowledge representation
- increasing depth of INFOSEC and environment knowledge bases
- configuring a security solution, taking into account which mechanisms address which threats. For this capability, the highest priority threat would be addressed by appropriate mechanisms, and then these mechanisms would be applied to the remaining list of threats, noting which ones were also addressed. These additional threats would not need further consideration.
- distinguishing between unclassified and classified users, and providing reports at appropriate clearance levels
- checking for inconsistencies in user answers
- allowing the user to re-prioritize threats before we begin analysis of mechanisms
- providing an "undo" capability, so a user can change an answer
- improving user interface[1]
- investigating numerous applications with other DoD organizations (e.g., other horizontal domains beyond INFOSEC: COMSEC, COMPSEC, etc.)

4.2 Extending the Gadfly Approach

The application of model-based expert system technology to configure a software system that ensures certain characteristics outside the software domain itself, such as security or other quality aspects, and the linking of independently developed models to accomplish this task, is an exciting development. This subsection presents the theory behind this type of extension to a Gadfly effort.

The Gadfly approach is an innovative technique that provides guidance to a developer who is concerned about building a system in a particular domain and engineering certain quality attributes into that system at the architectural stage. It works by combining separately-developed knowledge bases about the application domain and about the quality attributes of interest.

Software architecture is the focus of a flurry of recent interest in software engineering, the overall intent of which is to elevate it from the craft stage towards a more comprehensive engineering discipline. Two parallel investigations seek to exploit its potential, and the Gadfly approach discussed here resides at the intersection of these two investigations. The first, domain specific software architectures (DSSA),[2] is an attempt to engineer a family of related products,

1. Currently underway (under contract).

2. Section Appendix H provides a brief overview of DSSA.

all based on common architectural constructs. For example, codifying and supporting the architecture found in many avionics systems should enable more rapid and reliable production of future avionics systems which differ from their predecessors only in details and not in their fundamental composition. The second investigation is into the relationship of quality attributes (such as maintainability or security) and software architectures. In particular, what attributes must the architecture possess in order for the full system to achieve the desired qualities?

Viewed in the abstract, the UIA Gadfly approach produces an expert system whose questions and rules are a function of independent and separately-developed knowledge bases, one of which concerns software quality, the other of which concerns an application domain. In the two instantiations, the software qualities have been black-box testing and information security; the application domains have been abstract data types and command centers.

It is not difficult to imagine extending this scheme to include other quality attributes and application domains that are understood well enough that rules for application can be codified. The same schema can be used for other quality attributes as was used for information security:

- What are the possible consequences to the system if the desired software quality is not well enough engineered?
- What are the scenarios by which these consequences could occur?
- What objectives must we have in order to counter or preclude these consequences?
- What are the mechanisms necessary to realize the objectives?

Figure 4-1 shows the components of a generalized quality/domain Gadfly assistant. Knowledge bases on the left contain information corresponding to the above schema for different quality attributes. Knowledge bases on the right contain reusable architectures and composition rules about specific application domains.

Future extensions to the Gadfly approach would attempt to populate the schema (and hence propose embryonic knowledge bases) of, for example, the quality attributes of availability and performance.

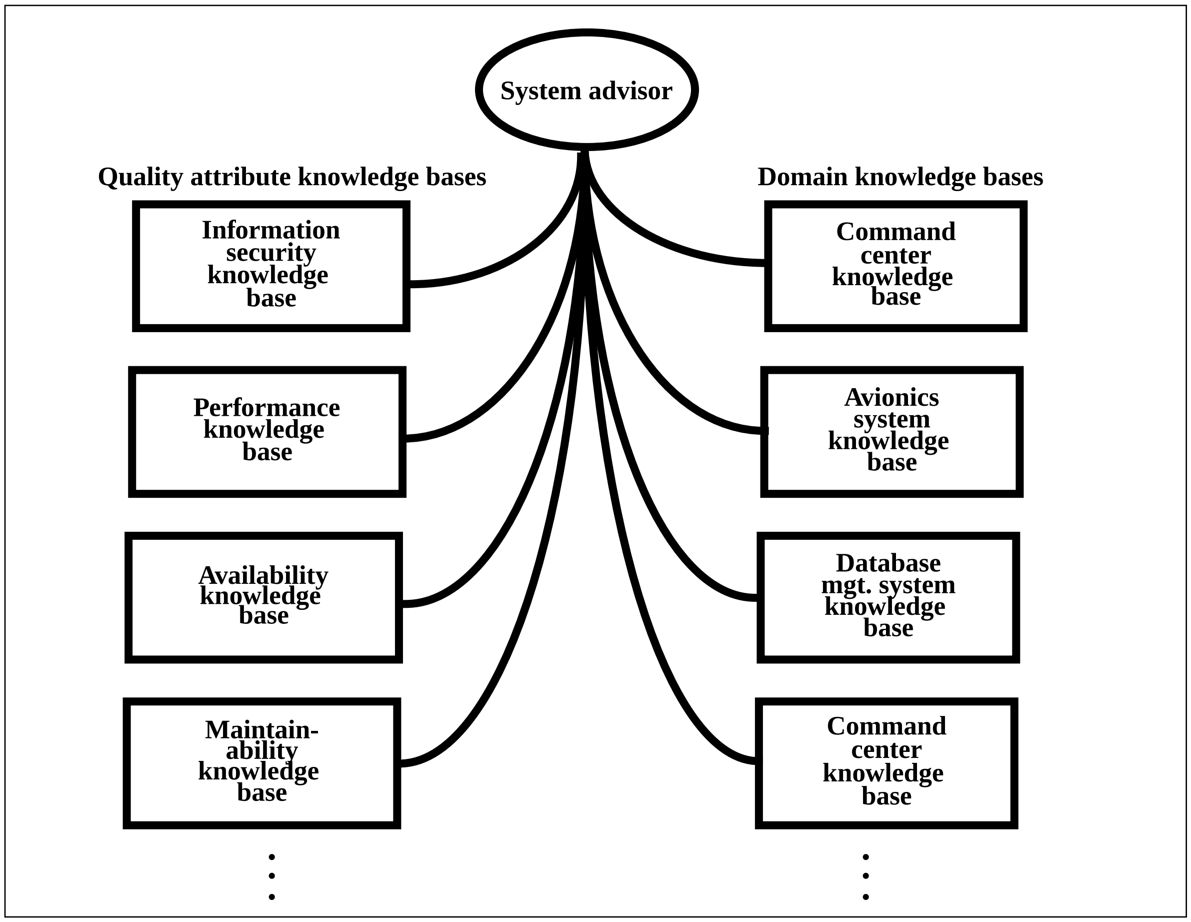

System advisor
Quality attribute knowledge bases
Domain knowledge bases
Information security knowledge base
Performance knowledge base
Availability knowledge base
Maintain-ability knowledge base
Command center knowledge base
Avionics system knowledge base
Database mgt. system knowledge base
Command center knowledge base

Figure 4-1: A Generalized Gadfly

5 Conclusions

CARDS and the SEI advocate a model-based approach to reuse. This approach is predicated on an understanding software reuse as derived from disciplined engineering practices. The feature of the engineering discipline that is of relevance to the following discussion is that such disciplines make use of a publicly-held collection of engineering models. These models are taught at the university level; are the basis for various engineering standards; and are an integral component of engineering problem-solving and design processes. It is the public disposition of these models, and the formal means of transmitting these models (e.g., through university training and handbooks) which distinguishes an engineering discipline from a craft.

A useful advisor/analyst application can be developed. Thus the basic concept has indeed been proven. Interest in the proof of concept demonstration has been strong, and it is recommended that development of a full prototype and system get underway. To improve the assistant, additional knowledge is needed to extend both the CARDS GCCL domain model and INFOSEC (UIA) models. While the knowledge is sufficient for a proof of concept, development of a full system will require additional modeling in both the software domain and the INFOSEC domain.

The Feature-Oriented Domain Analysis methodology introduced by SEI was helpful in extending the existing models and formulating the overall picture. In particular, it was helpful to add the knowledge about concepts of operation and environment. This information is lacking in the current INFOSEC and CARDS models, and is a critical part of the knowledge used by analysts in carrying out their activities. The proof of concept effort included developing a small part of this knowledge base in CLIPS rules, for rapid prototyping. A full prototype of Gadfly must include substantial additional information.

CLIPS rules were used to model the concepts of operation and environment information, as well as the links between software components and threats. Because rules are very flexible, can readily be developed incrementally, and have a low initial effort needed in the knowledge engineering, they provided the optimal way to develop a proof of concept. Now that the domain is better understood, it would be preferable to develop a more complete model in structured representation such as RLF. One aspect of developing the prototype should be the creation of an RLF model for this area to add/complement the UIA model.

Appendix A Milestone Chart

The following table provides the project milestones for the UIA Gadfly proof of concept.

Table A-1: UIA Gadfly Proof of Concept Milestone Chart

Task/Month	4/94	5/94	6/94	7/94	8/94	9/94	10/94	11/94	12/94
Task 1:									
Prelim UIA scenario		▲	▲						
Prelim Gadfly Design			▲						
Detailed Gadfly Design & Scenario			▲---	▲					
UIA Model Evolution		▲-	-----	-----	-----	---▲			
Develop & test UIA Gadfly				▲---	-----	---▲			
UIA Gadfly POC demo						▲			
Report and Recommenda-tions						▲--	----	----	---▲

Δ - planned start, planned finish

▲ - started, completed

Appendix B The CARDS System Composer

The CARDS system composer is a tool that permits reuse library users to compose alternative systems from components held within the library.[1] Valid compositions are described by a model of the application architecture, i.e., by a model of the major application components and connections among these components. The current operational system composer supports the packaging of composed systems for extraction from the reuse library; it also supports the creation of load images and interactive demonstration of composed systems. The system composer is described in detail in [CARDS 94]. The following description focuses on the major components (models and computational strategies) employed by the composer, and introduces the terminology used to describe these components.

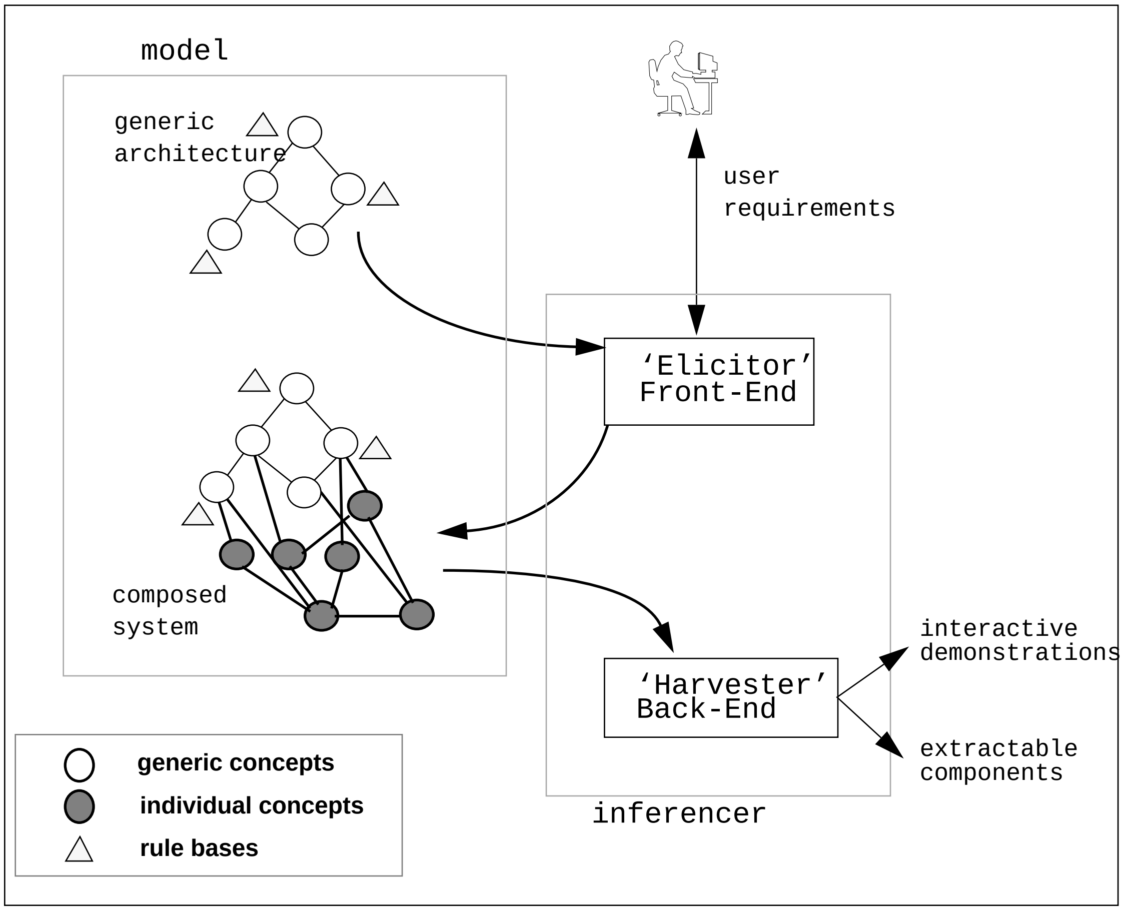


Figure B-1: Elements of the CARDS System Composer

1. A library is a model-based repository such as the CARDS Generic Command Center Library (GCCL).

Figure B-1 illustrates the major elements of a CARDS-style model-based reuse assistant. At the top level, there are models and inferencers. Models describe some kind of domain knowledge, which inferencers use to engage in dialogues with library users. The most central kind of knowledge model employed in CARDS libraries is the software architecture. We denote this as a knowledge model, rather than merely a system model, because the model describes not just a single system, but a family of systems that can be composed through a design refinement process. That is, the model contains a number of decision points which the user must pass to reach a solution; the process of successively traversing decision points leads to a gradually refined model, and ultimately results in the selection and composition of components. The refinement process is represented in the models through the individuation of a generic model, i.e., individual concepts and connections are created from a model of generic concepts and connections.

The inferencers perform the computational processes that drive the user dialogue and refinement. Inferencers are, essentially, intelligent agents that can traverse the models, interpret their meaning, and mediate a dialogue between the user and the model to achieve some end result. Inferencers can employ a variety of reasoning techniques and exploit the integration of production-style (i.e., rule-based) reasoning with structured reasoning. For example, a version of the Elicitor inferencer, depicted in Figure B-1, traverses the RLF semantic network[1] that encodes the structural relationships of the Air Force Generic Command Center Architecture (GCCA).[2] At various nodes of the structural model, the Elicitor will consult localized rule bases, and engage in a forward-chaining "**Think**-about-the-model, **Ask**-the-user, **Update**-the-model" (TAU) inference cycle. This inference cycle will in turn result in new questions to the user, result in changes to the semantic network (e.g., individuation of generic concepts), and, possibly, change the focus of the Elicitor to another node in the structural model where the TAU process will be repeated.

The composer represents just one kind of reuse service that can be centered on a formal model of an application architecture. The UIA provides additional opportunities for developing architecture-based reuse services. However, unlike the composer service, which provides a service based solely on an architectural model (e.g., the GCCA), the UIA service involves the integration of two models, i.e., integration of the UIA, for security properties, with some architectural model (or set of models) for a domain such as command centers.

This integration of, and reasoning from, separately modeled and maintained RLF knowledge bases was first developed for the Black Box Gadfly. Appendix C briefly describes this original Gadfly application. This application served as a starting point for the UIA Gadfly.

---

1. The knowledge encoded in the semantic network consists of both structural and rule-based models. The structural model provides an understanding of how elements fit together (i.e., the structure of the semantic network). The rule-based model provides an understanding of the issues and/or rules associated with the elements (i.e., the questions asked at each node of the semantic network)

2. The GCCA is implicitly captured in a domain model within the GCCL.

Appendix C The Black Box Gadfly Assistant

The first RLF-based reasoning assistant was named Gadfly.[1,2] Gadfly was a black-box test plan generator assistant that engaged an Ada programmer in a Socratic dialogue (i.e., it asked probing questions about the programmer's assumptions about operations (functions and procedures) that comprised a specific Ada package specification, and then generated black-box test plans to confirm these assumptions). For example, a procedure that has as an argument of type FILE_TYPE might trigger a sequence of questions concerning assumptions in the subprogram about file permissions, whether the file handle had been initialized prior to use, device capacity, etc. In each case, Gadfly would ask about these assumptions, then ask about the expected behavior of the program given a violation of these assumptions. Given assumptions and expected behavior, a black-box test plan would be generated.

The scenario is necessarily truncated and oversimplified. However, the underlying principle is that some level of expertise about the nature of black-box testing was encoded in a structured conceptual model. This structured model described various types of black-box testing approaches (e.g., error guessing, boundary testing, and robustness testing). Each type of testing would result in a different style of dialogue and, ultimately, a different test plan strategy. The black-box testing knowledge was then applied to a separately-developed model of Ada subprograms and packages. These models were separately developed since it was clear that black-box knowledge could be applied equally well to Ada, C, and a number of other formalisms.

Figure C-1 illustrates how Gadfly in effect integrated horizontal domain knowledge (black box testing) with vertical domain knowledge (Ada).[3] First, the generic model of Ada units is *individuated* to the specific unit under investigation. In the Gadfly prototype this individuation network was created automatically by an Ada parser. This individuation model was input to the Gadfly inferencer (not illustrated in Figure C-1); additional input comes from the user, as directed by the black-box testing model. For example, the black-box model would provide the user with a set of testing strategies from which the user would select; given a testing strategy, the Gadfly inferencer would traverse the appropriate, strategy-specific black-box test plan models, consulting the Ada unit model to ask questions appropriate to both the Ada unit and selected testing strategy. As a by-product of the dialogue, a specific test plan would be individuated, and linked to the Ada unit.

1. For the development of the Gadfly, Unisys was the prime contractor for the CARDS contract.

2. "Gadfly" was a derisive nickname given to Socrates by ancient Greeks who became frustrated (fatally so) with his knack of asking irritating but ultimately critical questions about prevailing philosophies.

3. Testing is considered horizontal because it would apply equally well to many different programming language formalisms; the Ada model is considered vertical in this context because it reflects a specific kind of programming formalism.

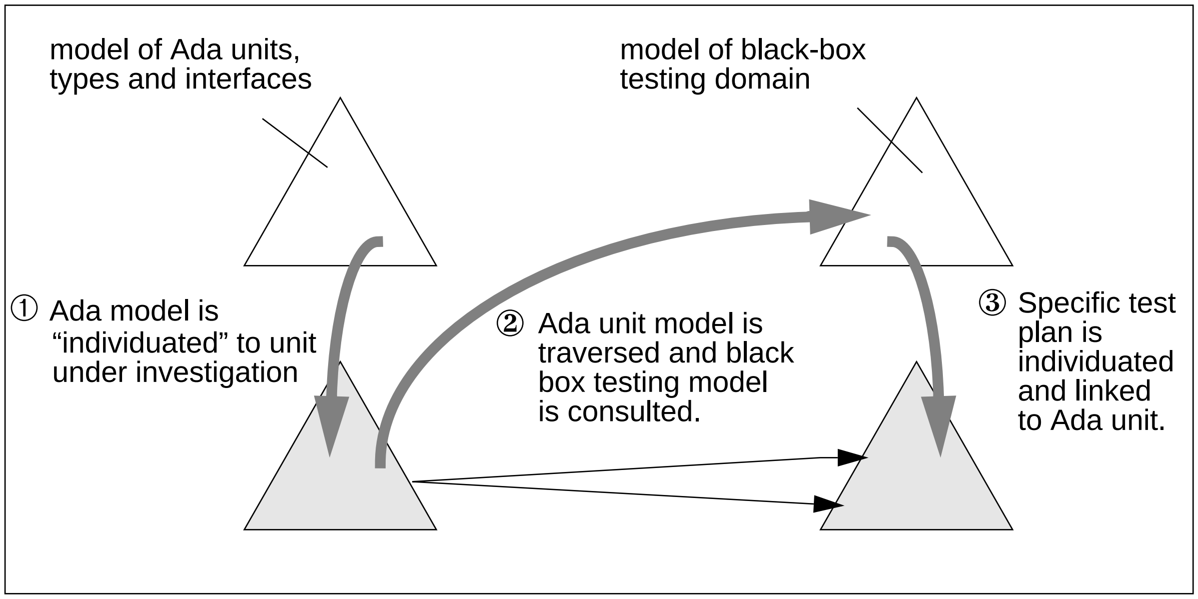

model of Ada units, types and interfaces
model of black-box testing domain
① Ada model is "individuated" to unit under investigation
② Ada unit model is traversed and black box testing model is consulted.
③ Specific test plan is individuated and linked to Ada unit.

Figure C-1: Integration of Horizontal and Vertical Domain Models

Gadfly could be used to generate test plans from already existing Ada specifications; it could also be used "in reverse;" that is, by programmers prior to the development of a specification (or implementation of a given specification). These two kinds of uses reflected different roles for such an assistant: as an aid in independent test and evaluation, and as a design aid. In the latter case, Gadfly could be used in the formulation of better-designed abstract data types.

Appendix D Feature-Oriented Domain Analysis

D.1 Introduction

The Software Engineering Institute (SEI) developed Feature-Oriented Domain Analysis (FODA) methodology resulting from an in-depth study of other domain analysis approaches. Successful applications of various methodologies pointed towards approaches that focused on the process and products of domain analysis. The FODA feasibility study [Kang 90] established methods for performing a domain analysis, described the products of the domain analysis process, and established the means to use these products for application development.

D.2 Foundations of the FODA Methodology

The FODA methodology was founded on a set of modeling concepts and primitives used to develop domain products that are generic and widely applicable within a domain. The basic modeling concepts are abstraction and refinement. Abstraction is used to create domain products from the specific applications in the domain. These generic domain products abstract the functionalities and designs of the applications in a domain. The generic nature of the domain products is created by abstracting away “factors” that make one application different from other related applications. The FODA method advocates that applications in the domain should be abstracted to the level at which no differences exist between the applications.

Refinements are used to both refine the generic domain products and to refine the domain products into applications. Once the abstraction of the applications in the application domain is completed, the factors that make each application unique are incorporated into the generic domain products as refinements of the abstractions. Specific applications in a domain may be developed as further refinements of the domain products by using the general abstraction as a baseline and selecting among alternatives and options to develop the application (i.e., those factors that have been abstracted away must be made specific and reintroduced).

Abstracting the applications in the application domain is accomplished by using the modeling primitives of: aggregation/decomposition, generalization/specialization, and parameterization. The FODA method applies the aggregation and generalization primitives to capture the commonalities of the applications in the domain in terms of abstractions. Differences between applications are captured in refinements. An abstraction can usually be refined (i.e., decomposed or specialized) in many different ways depending on the context in which the refinements are made. Parameters are defined to uniquely specify the context for each specific refinement. The result of this approach is a domain product consisting of a collection of abstractions and a series of refinements of each abstraction with parameterization. Understanding what differentiates applications in a domain is most critical since it is the basis for abstractions, refinements, and parameterization.

The feature-oriented concept of FODA is based on the emphasis placed by the method on identifying prominent or distinctive user-visible features within a class of related software systems. These features lead to the conceptualization of the set of products that define the domain.

The FODA methodology has been well defined and documented in both the window manager and movement control domains [Kang 90, Cohen 92, Krut 93]. Numerous documents have captured FODA's evolution conceptually and in the representation and use of the methodologies products. The following section provides a brief overview of the FODA process and the models that are produced from the process.

D.3 FODA Process and Products

The FODA feasibility study [Kang 90] defined a process for domain analysis and established specific products for later use. The basic phases that characterize the FODA process are:

1. Context Analysis, which defines the extent (or bounds) of a domain for analysis.
2. Domain Analysis, which provides a description of the problem space in the domain.

The following subsections provide a textual overview of the context analysis and domain modeling phases. Each of these analyses uses modeling techniques to create an abstract representation of the domain.

D.3.1 Context Analysis

Context analysis defines the scope of a domain that is likely to yield useful domain products. During the context analysis of a domain, the relationships between the "domain of interest" and the elements external to it are established and analyzed for variability. The kinds of variability to be accounted for are, for example, different data requirements and/or operating environments among applications in the domain. The results of the context analysis, along with other factors such as availability of domain expertise, domain data, and project constraints, are used to limit the scope of the domain.

The product resulting from the context analysis is the *context model*. This model includes a structure diagram and a context diagram (Figure D-1). The structure diagram for a domain is an informal block diagram in which the domain is placed relative to higher, lower, and peer level domains. Higher level domains are those of which the domain under analysis is a part of the domain to which it applies. Lower level domains (or subdomains) are those within the scope of the domain under analysis, but are well understood. Any other relevant domains (i.e., peer domains) must also be included in the diagram.

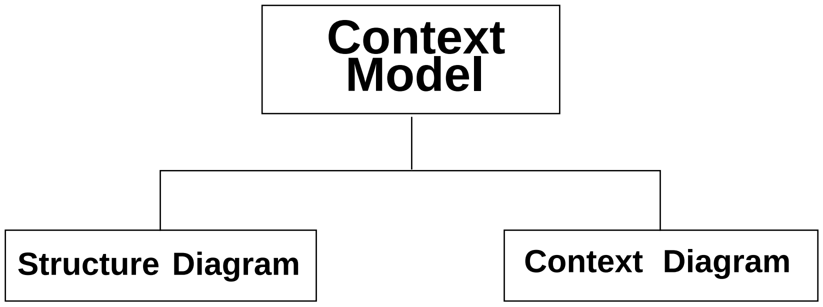


Figure D-1: Components of the Context Model

The context diagram is a data flow diagram showing data flows between a generalized application within the domain and the other entities and abstractions with which it communicates. One thing that differentiates the use of data flow diagrams in domain analysis from other typical uses is that the variability of the data flows across the domain boundary must be accounted for with either a set of diagrams or text describing the differences.

These products provide the domain analysis participants with a common understanding of

- the scope of the domain
- the relationship to other domains
- the inputs/outputs
- stored data requirements (at a high level) for the domain

D.3.2 Domain Analysis

Domain analysis identifies and models the commonalities and differences that characterize the applications within the domain. The product resulting from the domain analysis is the *domain model*. The domain analysis (or domain modeling phase) consists of three major activities. Figure D-2 illustrates the three components of a domain model.

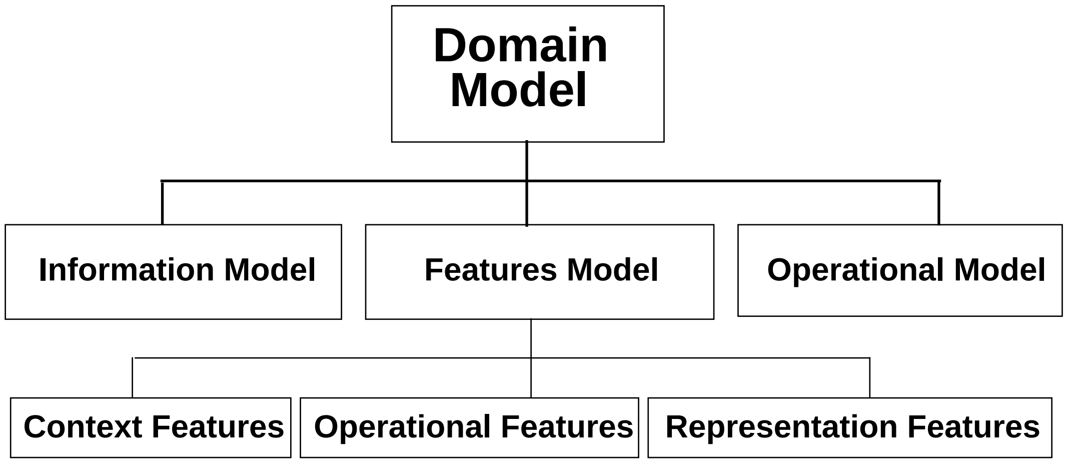


Figure D-2: An Illustration of the Three Components of the Domain Model

A brief description of each activity and its results are given below:

1. Information Analysis captures and defines the domain knowledge and data requirements that are essential for implementing applications in the domain. Domain knowledge typically is information deeply embedded in the software and it is often difficult to trace. Those who maintain or reuse software need this information in order to understand the problems the domain addresses.

   The information model may take the form of an entity-relationship (ER) model [Kang 90], a semantic network [Cohen 92], or other representations such as object modeling [Rumbaugh 91].

   The information model is used primarily by the requirements analyst and the software designer to ensure that the proper data abstractions and decompositions are used in the development of the system. The information model also defines data that is assumed to come from external sources.

2. Features Analysis captures a customer's or end user's understanding of the general capabilities of applications in a domain.[1] For a domain, the commonalities and differences among related systems of interest were designated as features and are depicted in the features model. These features, which describe the context of domain applications; the needed operations and their attributes; and representation variations, are important results because the features model generalizes and parameterizes the other models produced in this domain analysis.

   Features in the features model may be defined as alternative, optional, or mandatory. The mandatory features represent the baseline features of an application and the relationships between those features. The alternative and optional features represent the specialization of more general features.

   The features model development initially partitions features into context, representation, and operational features.

   - The context features are those that describe the overall mission or usage patterns of an application.
   - The representation features are those features that describe how information is viewed by a user or produced for another application (i.e., what sort of input and output capabilities are available).
   - The operational features are those features that describe the active functions carried out (i.e.,what the application does).

   The features model is the chief means of communication between the customers and the developers of new applications. The features are meaningful to the end users and can assist the requirements analysts in the derivation of a system specification that will provide the desired capabilities. The features model provides end users with a complete and consistent view of the domain.

3. *Operational Analysis* identifies the control and data flow commonalities and differences of the applications in a domain. This activity abstracts and then structures the common functions found in the domain and the sequencing of those actions into a model. Common features and information model entities form the basis for the abstract operational model. The control and data flow of an individual application can be instantiated or derived from the operational model with appropriate adaptation.

   The operational model is the foundation upon which the software designer begins the process of understanding how to provide the features and make use of the entities selected.

The domain modeling process also produces an extensive *Domain Dictionary* of terms and/or abbreviations that are used to describe the features and entities in the model and a textual description of the features and entities themselves.

1. A user may be a human user or another system with which applications in a domain typically interact.

The domain dictionary has been one of the most useful products of a domain analysis. The dictionary helps to alleviate a great deal of miscommunication by providing the domain information users with

- a central location to look for terms and abbreviations that are completely new to them
- definitions of terms that are used differently or in a very specific way within the domain

D.4 Applying the Results of Domain Analysis

FODA defines a method for performing domain analysis and describes the products (context models, domain models) of an analysis. A system developer works with the domain analyst and these products to define requirements for a system. The three steps in the process are

1. The developer and domain analyst use the features model as a vehicle for communicating system needs. The domain analyst will turn these needs into a selection of features. In addition, composition rules among features will automatically add specific features to the new system.
2. The domain analyst uses the information model to explain the *objects* that comprise a system. This helps the system developer understand the data requirements and other systems and data structures with which the system must interoperate.
3. The operational model is then used to describe commonality and differences in data and control flow resulting from differing combinations of features.

The operational model supports feature selection as well as architectural development. Feature selection will parameterize the operational model, thus establishing the dynamics of interacting system capabilities. A system developer will utilize this information to make choices that will affect both system control and operations. For example, a choice of features may affect the sequence of operations or eliminate those operations altogether. Another important aspect of this model is the definition of data flow resulting from these operations. The system dynamics necessary to meet the desired system capabilities may depend on specific feature selections.

Appendix E Examples of FODA Models for Security Analysis

E.1 Overview

This appendix provides rationale for and examples of domain analysis for the activity of the DoD branch (the *organization*) providing *security analysis* of information systems under the INFOSEC security model. The artifact of discussion is a *scenario*, under which a subscriber solicits the help of the organization to design or field a system that satisfies its security requirements, or solicits the help of the organization to assess the security fidelity of an already-designed or -fielded information system.

The goal of this appendix is to provide domain analysis examples than can be reviewed by INFOSEC domain experts to produce a complete domain analysis of INFOSEC system design and security analysis activities. In turn, the goals of that domain analysis are

- Support organizational activities to evaluate, recommend, and/or provide solutions to security deficiencies in candidate systems.
- Provide a common language for recommendation, correction, and evaluation activities.
- Build a capability in the evaluation/recommend activities to combine services and mechanisms to fulfill mission behavior.

E.1.1 Identifying the Domain: Organizational Missions, Functions, and Assets

Figure E-1 establishes the domain of interest by delineating the goals and missions of the organization, the functions that the organization performs in order to satisfy those goals and missions, and the assets that the organization can use to perform the functions.

E.1.2 Taxonomy of Scenarios

The scenarios under which security analysis is performed may be categorized by the following criteria:

- The stage that the subject system is in when the organization becomes involved in its development.
- Whether or not security has been considered in the design and development of the system to date.
- The role of the organization when becoming involved with the system: what result is being attempted?

These three criteria, shown in Figure E-2, define an enumeration of ordered triples that together characterize all of the scenarios under which the organization is asked to participate and assist in providing security to information systems. Not all triples are valid; for instance, the organization will never be asked to “correct” the security design of a “new” system, which by definition has no security design. Hence, no triple of the form (New, Correct, *)[1] represents a valid scenario.

1. “*” refers to any option. For example, (New, Correct, *) means (New, Correct, Yes) or (New, Correct, No).

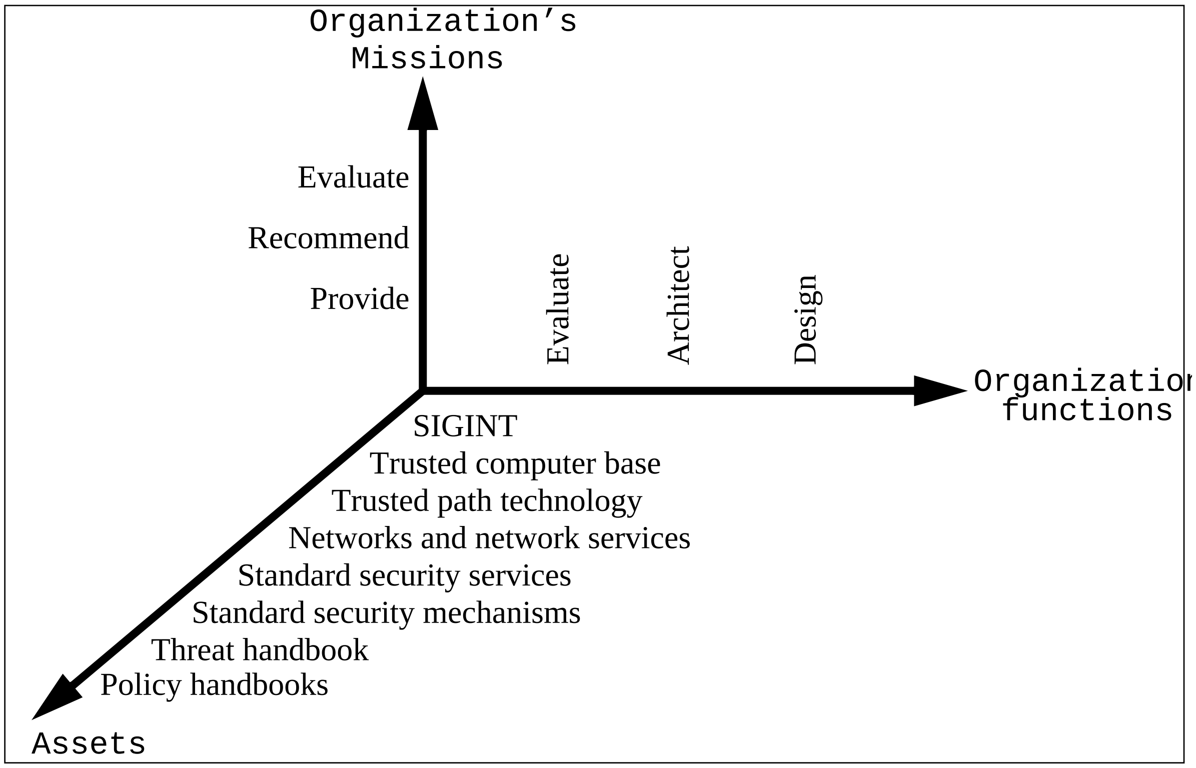


Figure E-1: Identifying the Domain: Organizational Missions, Functions, and Assets to Perform Security Analysis

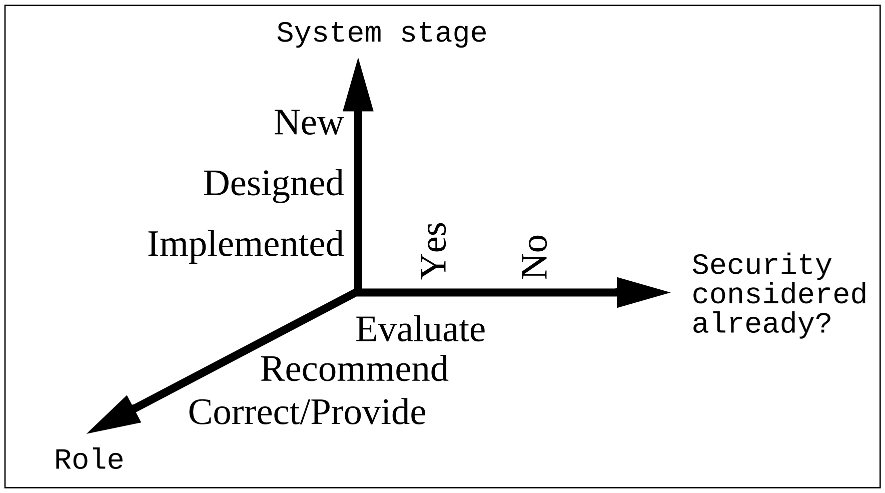


Figure E-2: Characterizing Security Analysis Scenarios

E.1.2.1 Roles played by the organization

The three roles that the organization takes on in the scenarios are as follows:

- Evaluation: Either (1) compare security properties of a system against criteria, including a threat analysis, or (2) determine if behavior of the system is compromisable.
- Recommendation: Suggest a range of options or solutions for design refinement.
- Correct/Provide: Refine and deliver an implementation of a solution.

Sometimes a recommendation may be requested, but cannot be provided because a specific recommendation might reveal classified approaches or techniques used to compromise the system. In such cases, the organization will provide an evaluation of the system instead.

E.1.2.2 System stages

A system brought to the organization for scrutiny will be in one of three stages:

- New: The development is beginning with a problem definition. No design or implementation work has been performed.
- Designed: Some or all design has been done. This is evidenced by documentation such as a refined performance specification, or perhaps complete component specifications.
- Implemented: The system exists, and is now to be upgraded.

E.1.3 Leading Agents in the Scenarios

The type of individual agents involved for each security analysis scenario are given in the table below. The agent roles are:

- SA: Security architect. This is a systems engineer with security expertise.
- DD: Detailed designer. This person is a hardware or software engineer with security experience.
- EV: Evaluator. This person has a range of expertise, including communications and computer security.

Table E-1: Lead Agents in Scenarios, as a Function of System State and Organization's Role

Type of system	Evaluation	Recommend	Correct/ Provide
New	SA	SA	SA/DD
Designed	EV	SA	DD
Implemented	EV	DD	DD

E.1.4 Type of System

The type of system also plays an important role in determining the course of the scenario. For example, knowing that the system is a command and control system establishes a basic context for the questions that the agents ask, background assumptions they make, and the segment of the security infrastructure (e.g., communications security) that they primarily bring to bear on the problem.

E.2 Examples of FODA Models

The domain analysis methodology used to capture security analysis information during the workshop was FODA. The FODA methodology is founded on a set of modeling concepts and primitives (Appendix D). These concepts and principles are used to develop domain products that are generic and widely applicable within a domain. As part of the "FODA Models for INFOSEC System Design" workshop, examples of domain products were generated that captured the INFOSEC system design and security analysis activities. During the FODA workshop, examples were generated for both *Scoping* the domain and *Understanding* the domain, respectively. This is represented by Figure E-3.

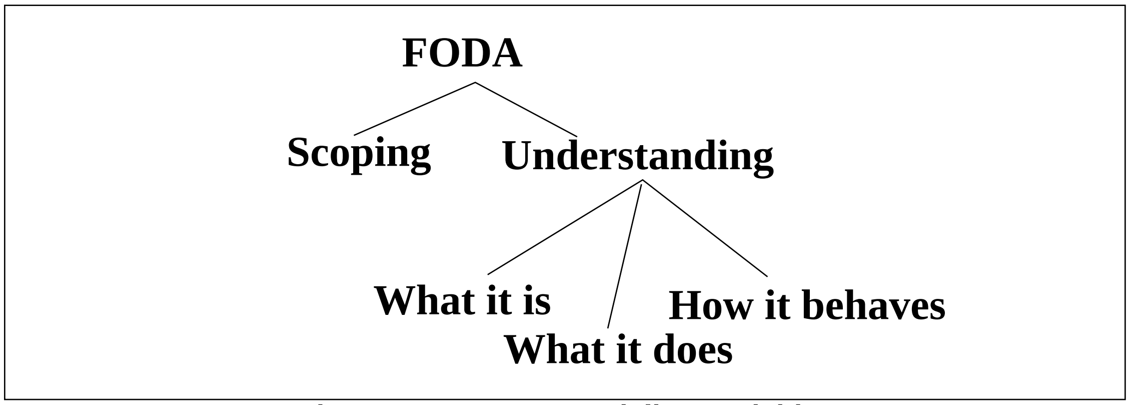


Figure E-3: FODA Modeling Activities

E.2.1 Scoping the Domain

The following subsections each provide an example(s) of models generated in scoping and understanding the domain. Appendix D provides the definitions and usage of each of these models.

Context analysis defines the scope of a domain. The product resulting from the context analysis is the *context model*. This model includes a context diagram and a structure diagram. These products provide the domain analysis participants with a common understanding of

- the scope of the domain
- the relationship to other domains
- the inputs/outputs to and from systems in the domain
- stored data requirements (at a high level) for the domain

The following figure provides an example of a context diagram for Security Analysis. Its inputs include information about the system to analyzed (mission constraints, threats), the intent of the effort (problem definition), and ongoing infrastructure constraints (policy). Its outputs include information about security recommendations and relative risks.

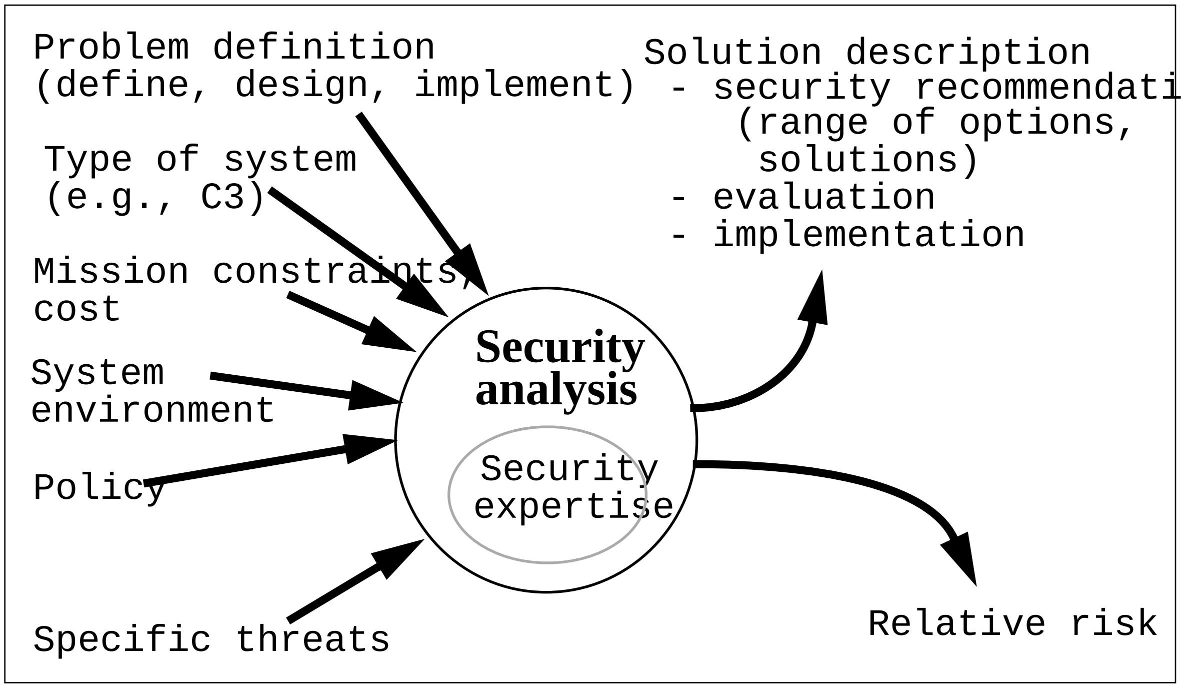


Figure E-4: Context Diagram for Security Analysis Activities

The following figure provides an example of a structure diagram for security analysis. Security analysis may be compared with safety analysis, fault tolerance analysis, and similar activities that evaluate a system with respect to one or more quality attributes. It relies on certain areas of experience, expertise, and analysis of similar problems.

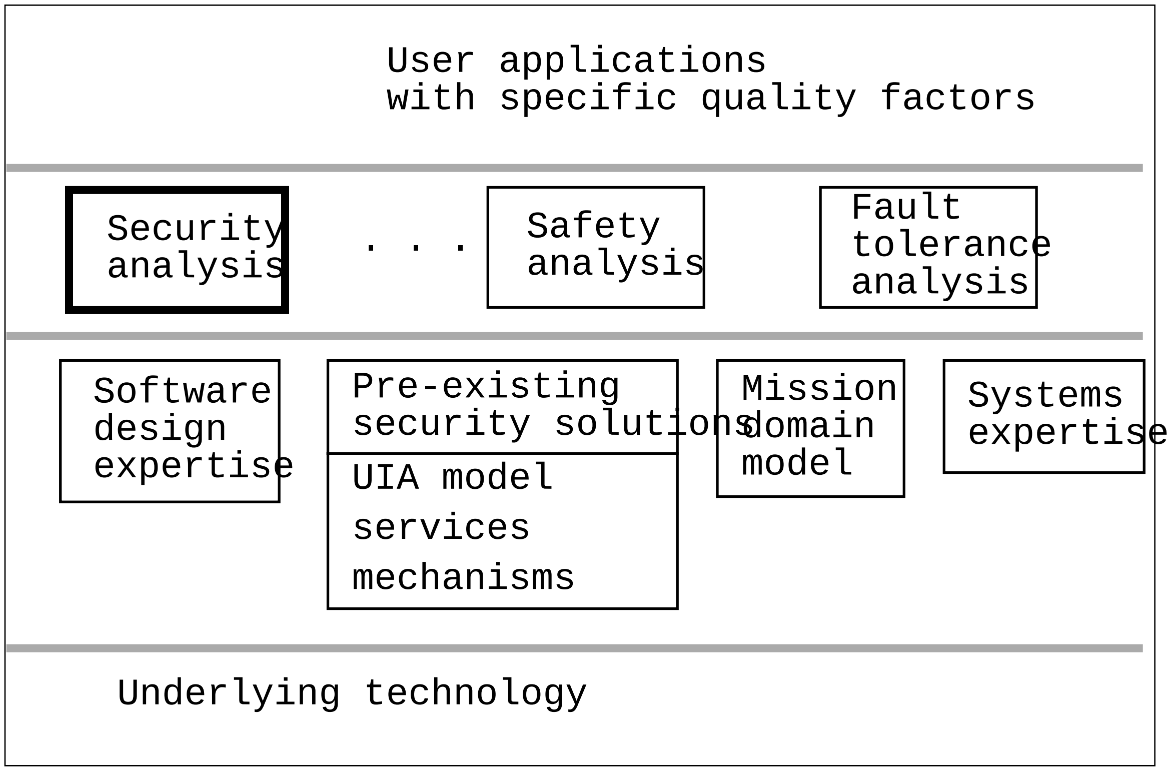


Figure E-5: Structure Diagram for Security Analysis Activity

E.2.2 Understanding the Domain

Domain analysis identifies and models the commonalities and differences that characterize the applications within the domain. The product resulting from the domain analysis is the domain model, which consists of three components. The information model represents what the applications are in terms of the entities; the features model captures what the applications do, both in terms of operations and context; and the operational model relates the information model and features model to the behavior and function of the applications.

The information model captures and defines the domain knowledge and data requirements that are essential for implementing applications in the domain. The information model may take the form of an entity-relationship (ER) model, a semantic network, or other representations such as object modeling. The example information model in Figure E-6 is an ER diagram. This diagram decomposes the security analysis domain into entities of threats, services, mechanisms, etc. In this figure, threats could be further decomposed (populated) to include information about the threat consequences and security objectives defined in Appendix F.

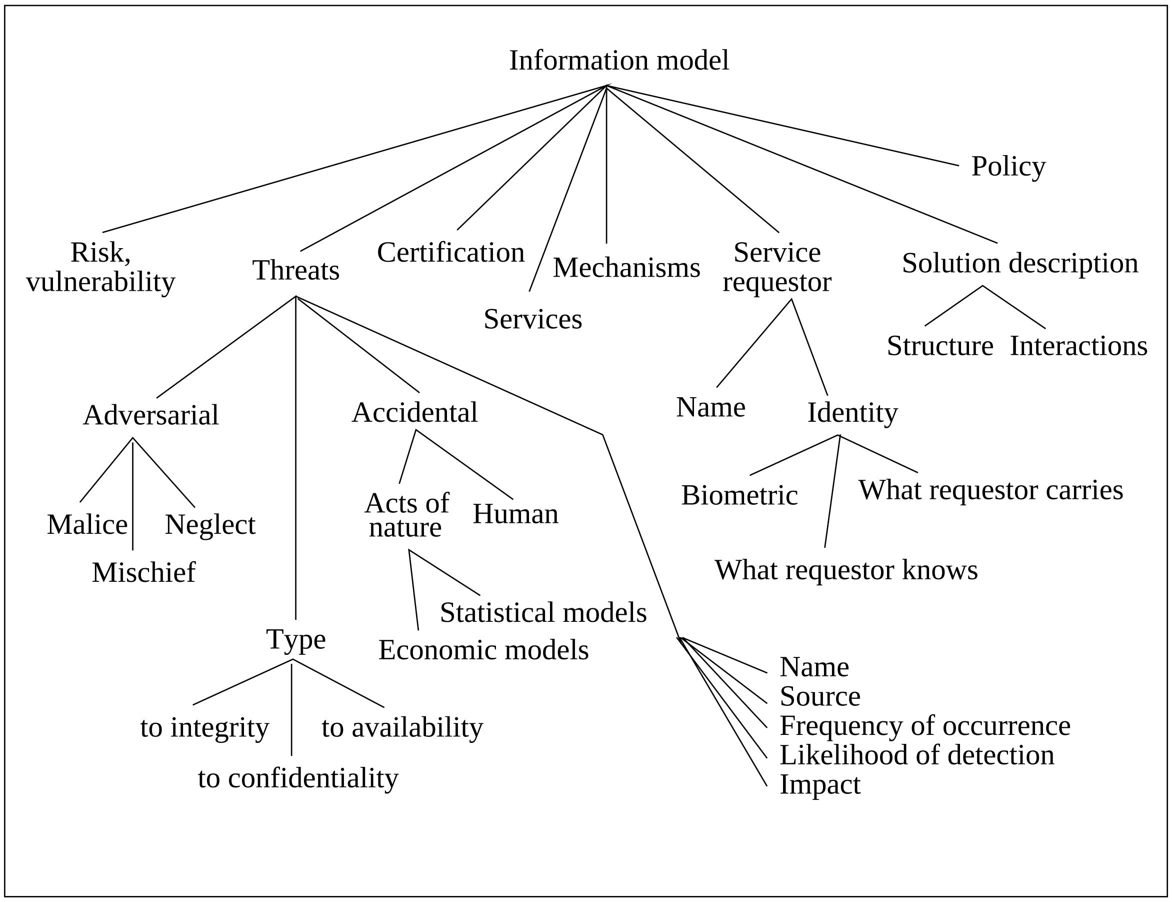


Figure E-6: Information Model for Security Analysis

The features model captures a customer's or end user's understanding of the general capabilities of applications in a domain. The features model partitions features into context, representation, and operational features:

- The context features are those that describe the overall mission or usage patterns of an application.
- The operational features are those features that describe the active functions carried out (i.e.,what the application does).
- The representation features are those features that describe how information is viewed by a user or produced for another application (i.e., what sort of input and output capabilities are available). [1]

Figure E-7 is an example of a context features model for security analysis. Each of these features address ways in which a security analyst would "look at" or use an application. Figure E-8 is an example of an operational features model for security analysis. This model helps its users understand that

- The threat analysis could consist of anything from calculating the risk to defining methods to over come the risk.
- The solution proposal may consist of evaluating, recommending, and/or providing.

Each of these operational features correlates to functionality and behavior that exists in some or all of the applications in a domain.

1. The representation features model was not covered in the workshop.

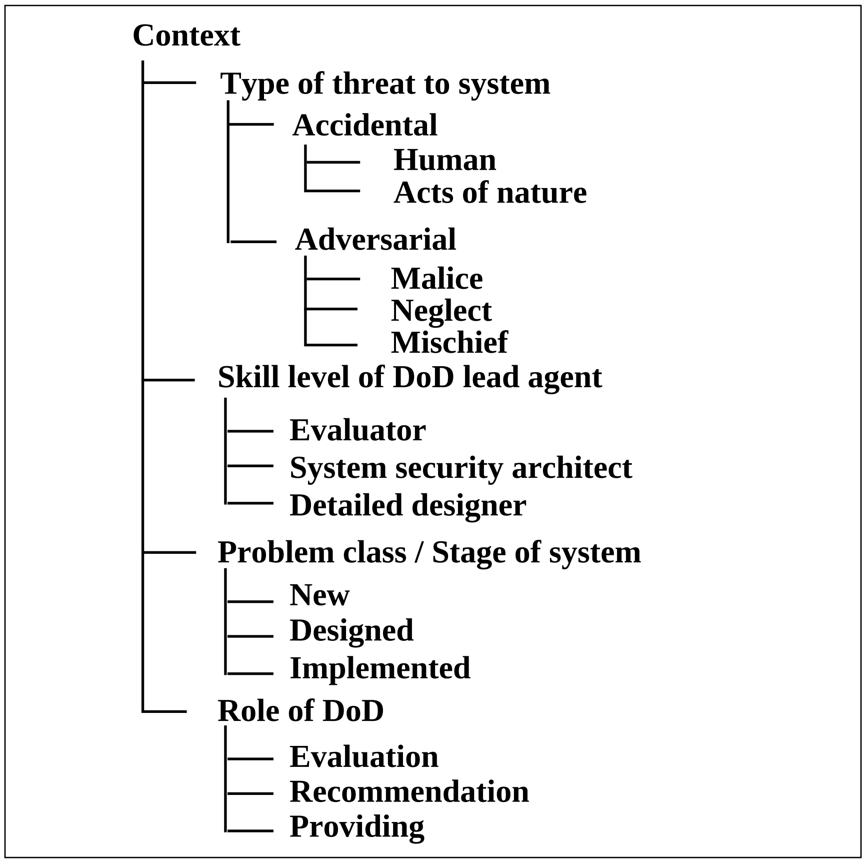


Figure E-7: Context Features Model for Security Analysis

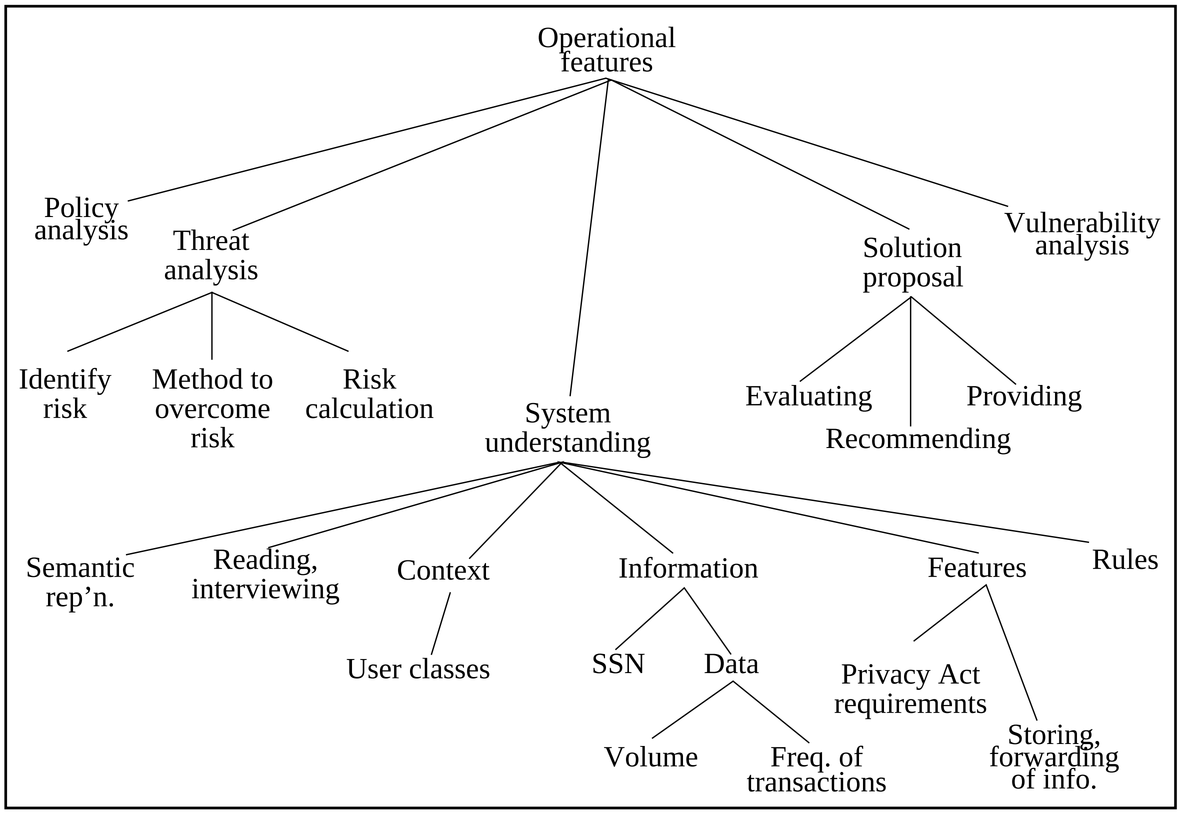


Figure E-8: Operational Features Model for Security Analysis

The operational model identifies the control and data flow commonalities and differences of the applications in a domain. The development of an operational model abstracts and then structures the common functions found in the domain and then sequences of those actions into a model. Common features and information model entities form the basis for the abstract operational model. The control and data flow of an individual application can be instantiated or derived from the operational model with appropriate adaptation.

The example operational model in Figure E-9 represents a high-level operational view of the organization’s activities when it is asked to design a new system. Figure E-9 represents the sequence of steps (or the security analysis process), and the inputs and products of each step. The model is quite similar for the (design, new, *) scenario and the (provide, new, *) scenario.

The operational model for the (evaluate, designed, *) scenario is also similar, except that understanding the system would then require understanding its given design.

The operational models for the (*, implemented, *) scenarios are yet to be determined.

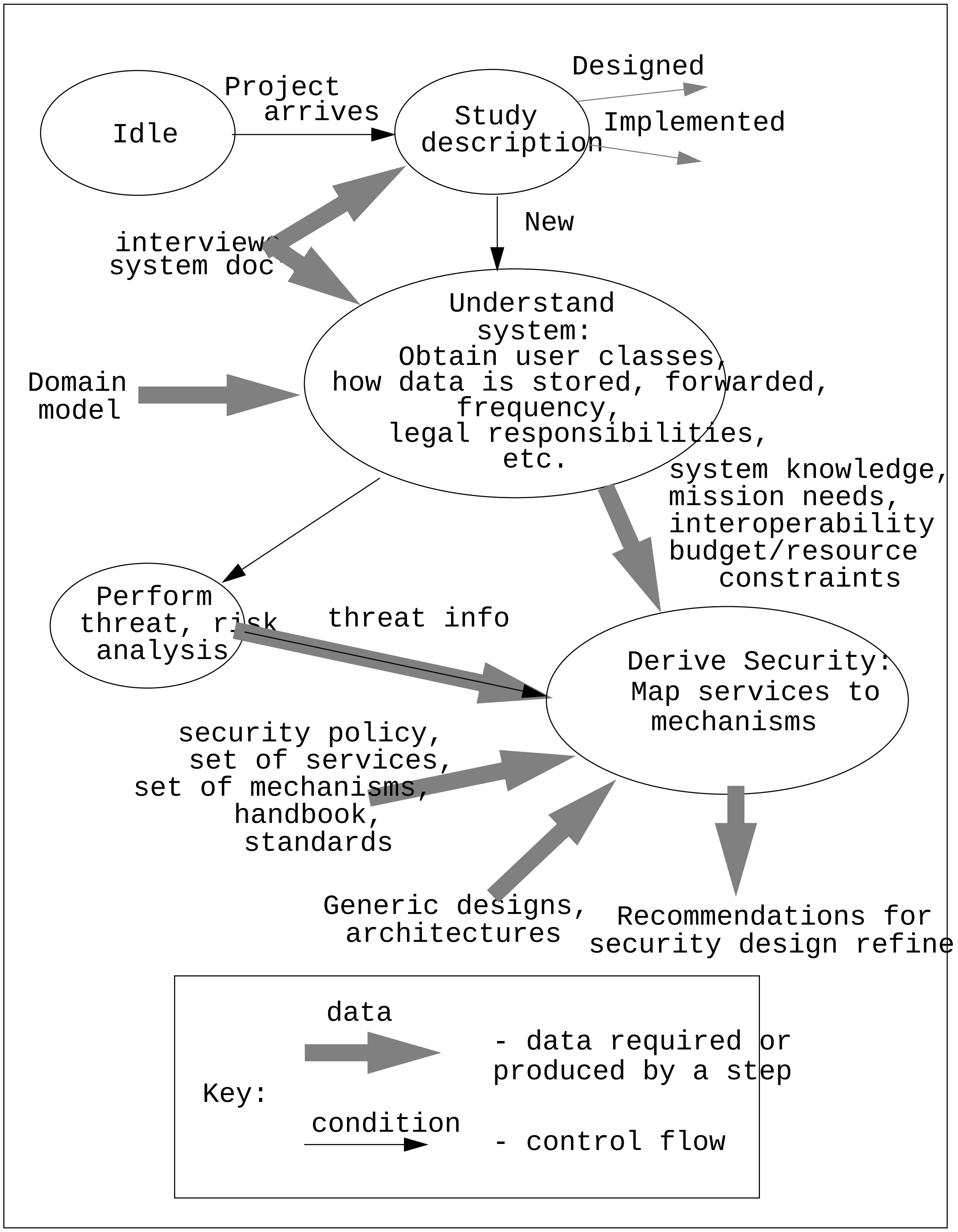

Idle
Project arrives
Study description
Designed
Implemented
New
interviews, system doc
Understand system: Obtain user classes, how data is stored, forwarded, frequency, legal responsibilities, etc.
Domain model
system knowledge, mission needs, interoperability budget/resource constraints
Perform threat, risk analysis
threat info
Derive Security: Map services to mechanisms
security policy, set of services, set of mechanisms, handbook, standards
Generic designs, architectures
Recommendations for security design refine
Key:
data
- data required or produced by a step
condition
- control flow

Figure E-9: Operational Model for Security Analysis of a New System

Appendix F UIA Topic Areas

The following provides examples of information that was used to formed the basis for the rules that tied the UIA model to the concept of operations model (i.e., environmental context knowledge model). In the UIA Gadfly, the information security knowledge base was structured in the following fashion:

Table F-1: Structure of Information Security Knowledge Base

	Threats	Services
Most general	**Threat consequence**: the negative consequence that a threat may have on the secure operation of an information system	**Security objective**: a countering force to a specific threat consequence
	General threat action: a group of related scenarios, each of which could cause the associated threat consequence	**Security service**: a type of protection that may be provided by an information system to enhance the system's ability to securely satisfy its intended mission
Most specific	**Specific threat action**: an individual scenario	**Security technique**: an implementation technique that provides a security service

This structure follows a defined terminology and model of INFOSEC threats and services [Gulachenski 94]. Threat consequences are the end result (to a computer/information system) of any threat, general or specific. Threat consequences "define the negative affect that a threat may have on the secure operation of an information system." The major threat consequences (disclosure, deception, disruption, usurpation) are respectively countered by the four major security services (confidentiality, integrity, availability, authenticity). These security services are described as principal security services. Each principal security service is defined by a protection objective, subordinate security services that refine and clarify the objectives, and potential implementation techniques. Table F-2 catalogs the threat consequences and threat actions; Table F-3 describes the countering principal security service objectives, services, and techniques.

Table F-2: Threats and Threat Consequences

Threat consequences	General threat actions	Specific threat actions
Disclosure	Exposure	Hardware/software errors Human errors Deliberate disclosure Scavenging
	Interception	Wiretapping Eavesdropping Emanations analysis
	Inference	Signals intelligence Traffic analysis Database query analysis
	Intrusion	Trespass Penetration Reverse engineering Cryptanalysis
Deception	Masquerade	Spoof Malicious logic
	Falsification	Substitution Insertion
	Repudiation	False denial of origin False denial of receipt
Disruption	Incapacitation	Malicious logic Physical destruction Hardware/software errors Human errors Natural disaster
	Corruption	Hardware/software errors Human errors Tampering Malicious logic
	Obstruction	Interference Overload

Table F-2: Threats and Threat Consequences

Threat consequences	General threat actions	Specific threat actions
Usurpation	Misappropriation	Theft of service Theft of functionality Theft of data
	Misuse	Malicious logic Tampering Violation of permissions

Table F-3: Security Objectives, Services, and Techniques

Security objectives	Security services	Techniques
Confidentiality	Data Emanations Traffic Signals	Access control Object reuse Encryption TEMPEST techniques Separation of components Administrative procedures
Integrity	System Data	Access control Checksums Digital signatures Recovery mechanisms Non-volatile memory Deterrence Configuration control Secure maintenance of components Inspection of hardware/software/firmware Comparison with known correct components
Availability	Resistance Recovery	Physical access controls Redundancy Information backup Dispersion Anti-tamper mechanisms Anti-jam mechanisms Hardening (EMP) Modular components Survivable design Mobility Operations security
Authenticity	Authentication Nonrepudiation	Physical access controls Biometric devices User login via ID/password Digital signatures Other crypto techniques

Appendix G Information Security (INFOSEC) Demonstration Script

G.1 Introduction

The purpose of this demo is to show how software tools can aid System Security Architects in the process of generating candidate architectures that provide Information Security (INFOSEC) solutions to reusable software components.

In particular, models of Information Security and software architectures can be tied to a "context model" to help System Security Architects analyze threats to a system or component and get recommendations on

- Missing threat information they should be concerned about (but did not specify).
- Lists of primary and supporting security "services" to counter threats.
- Discovery of the complex 'interrelationships' between various threats and services, and when they can/should be combined. The ultimate goal is to form lattices of primary and supporting security services to counter all threats.

G.2 Terminology

This demo follows the MITRE terminology and model of INFOSEC "threats" and "services." "Threats" are broken down as follows:

THREAT CONSEQUENCE --> GENERAL THREAT -> SPECIFIC THREAT

Threat "consequences" are the end result (to a computer/information system) of any threat, general or specific. "Services" are functions that counter threats. Services can be of type 'primary' or 'supporting,' where a primary service may or may not need a supporting service to ensure complete threat countering ability.

The original idea was to present the system with a scenario centered around the Drug Enforcement Agency (DEA) and its efforts to conduct quick, efficient raids on various individuals/organizations to combat the drug trade. For this, a command and control system is used; this system must stand up to various security threats, since many drug groups are highly sophisticated. The demo then, proceeds with the idea that a mobile group will conduct a raid and need a Command/Control system (via mobile phone, etc.) to receive instructions from the main site. The CARDS 'message_processing' component could be used, *if* the threats to it are considered acceptable for such a raid. This demo shows how to analyze threats in such as context.

G.3 Structural View of the Demo

This demo has four major parts. The user must specify

1. Which threat(s) are most important to safeguard the *entire* information security system and/or (reusable) software component they have chosen. (For this step, the INFOSEC LMDL model is queried. Questions presented are generated from this model.)
2. In what "context" or operating environment will the component/system operate, i.e., are guards present, etc. (For this step, a Knowledge Base of CLIPS questions is dynamically loaded, along with inference rules coupled with this "context" model.)
3. A particular configuration of the software component chosen.The component can be tailored in several forms, each with different threat priorities importance. (For this step, the component's LMDL model is queried to generate questions.)
4. A report is generated in the end, detailing:

   a. What the user specified in each of the three steps above (a reminder).

   b. Sets of new, "inferred" threats that the user *should* have specified in Step 1 but did not. This helps fill in holes in the user's choice of threats/priorities.

   c. A set of "primary services" that helps counter threats discovered.

   d. A set of "supporting services" needed when a primary service has been inferred.

G.4 The Demo Script

The INFOSEC analysis system has three major components:

- Specify system wide threats of most concern.
- Specify context/concept of operations under which the system will operate.
- Choose a particular configuration of the current component.

Once all three of these components have been executed, the system will generate a report. This script shows one path through the components and report:

1. Invoke the CARDS launcher screen and move to the "message_processing" node at the bottom.
2. Choose "message_processing," "Launch_Message_Processing_Model," and "OK."
3. Choose "Search" and enter "message_processing." Click on it, then "apply."
4. Choose "message_processing," then "Perform Action," then "InfoSec System."

5. Press [Return] at the introduction screen.

   Note: The Knowledge Base choice screen will appear.

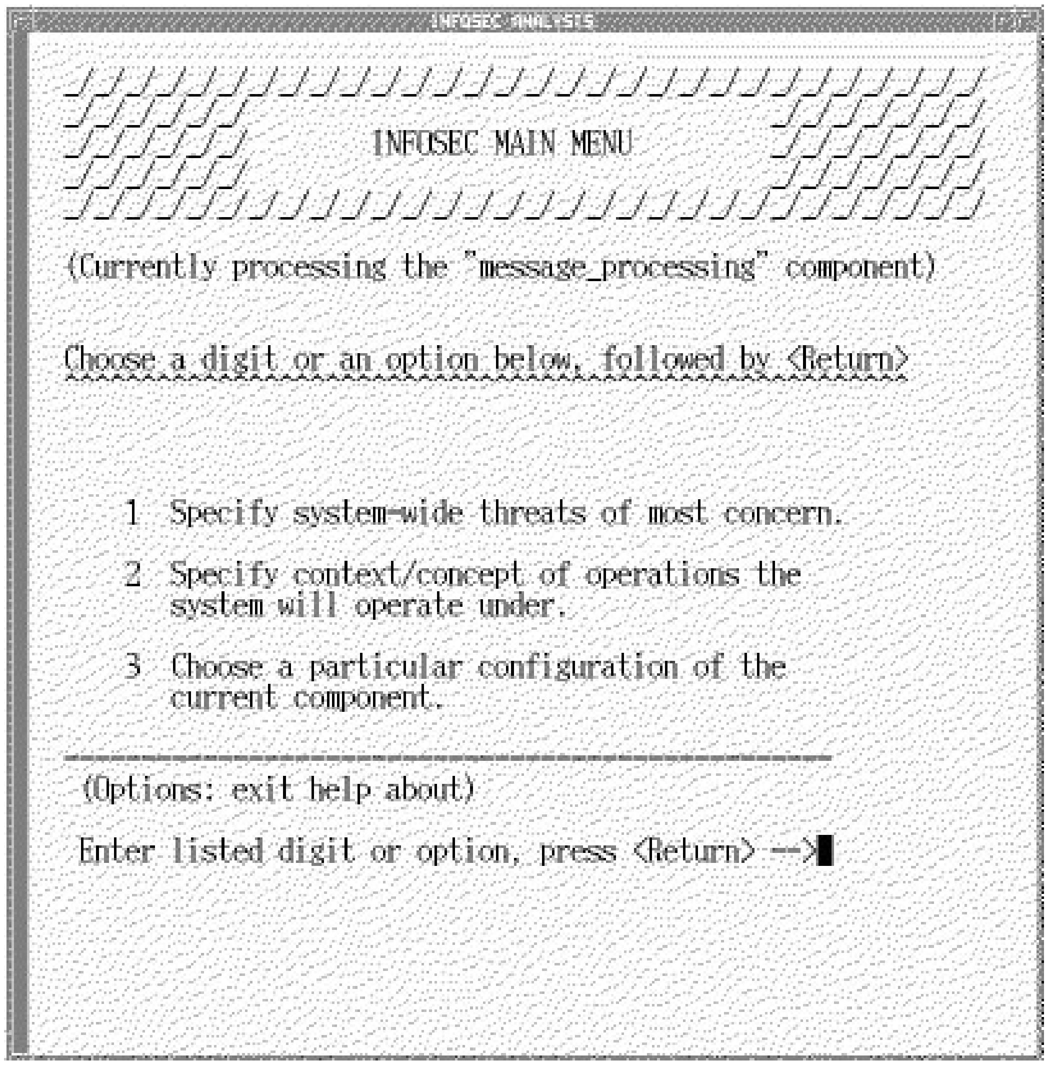
The demo is currently capable of running 2 knowledge bases. To see what each does, choose “help” or enter the choice at the prompt. Separate knowledge bases (KBs) allow the demo to analyze threats to the same reusable component (i.e, message_processing system) in various “contexts,” or modes of operations. Each KB represents a different set of circumstances that may generate an entirely different set of threats/services that the Security Architect should be concerned about when integrating this component into the entire INFOSEC system.

For now, we’ll choose the most recent/updated KB, DoD_KB_Ver_1.1.

6. Choose “DoD_KB_Ver_1.1” and press Return.

You’ll see the system dynamically loading in KB files, and querying the INFOSEC Library Model Description Language (LMDL) model for the names of the threat consequences and specific threats. The system is dynamically building a set of questions to ask the user regarding consequences/threats to the current component.

The “Main Menu” will appear (see Figure G-1).



```
                          INFOSEC ANALYSIS

                     INFOSEC MAIN MENU

(Currently processing the "message_processing" component)

Choose a digit or an option below, followed by <Return>


   1  Specify system-wide threats of most concern.

   2  Specify context/concept of operations the
      system will operate under.

   3  Choose a particular configuration of the
      current component.
______________________________________________________
 (Options: exit help about)

 Enter listed digit or option, press <Return> -->
```

Figure G-1: INFOSEC Analysis Main Menu

Even though we can choose these steps in any order, let’s do Step 1 first, ‘Specify system-wide threats of most concern.’ This step asks us to specify which consequences and specific threats are most relevant to counter for our *entire* INFOSEC system, as well as for the component chosen. We must also specify *how* important it is to counter each threat (on a scale of 1 to 3, where 3 is highest) because some threats are often far more important than others to counter.

7. Choose 1 ‘Specify system-wide threats of most concern.’

Say we are concerned about 3 main threat consequences -- Deception, Disclosure (of information) and Disruption (of our system). Just assume we are not all that worried about somebody ‘usurping’ our system.

8. Choose the threats by entering 1, 4, 7, 10, then press Return. Where 1,4, 7 and 10 are
   1 deception via falsification
   4 disclosure via exposure
   7 disclosure via intrusion
   10 disruption via obstruction

Assume we are really worried about disruption/interference, so we will choose number 1 (we're worried about excessive noise on our communication cables interfering with the system).

9. Choose 1 for '1 interference -- high threat.'

Assume we area bit concerned about system overloading and reverse engineering from an outsider.

10. Choose 6 and 8 for
    6 penetration — low threat
    8 reverse engineering— medium threat

We are worried about somebody within our group *deliberately* disclosing information, since this project has high visibility, and we are also worried that somebody will "scavenge" the system looking for tidbits of information.

11. Choose 1 and 10 for
    1 deliberate disclosure — high threat
    10 scavenging — high threat

And finally we are concerned somebody will "insert" false code/data into our system and/or "substitute" code/data in to deceive us in our operation.

12. Choose 1 and 6 for
    1 insertion — high threat
    6 substitution — low threat

    Note: The Main Menu will reappear.

We see that Step 1 has been completed, and we should now go to Step 2 to specify how and in what context our software system/component will operate.

13. Choose 2 'Specify context/concept of operations the system will operate under.'

We need to have one outside site in our system with which to communicate.

14. Choose 2 for 'Yes -- 1 site.'

Our external site will take commands from us.

15. Choose 2 for 'No -- sites(s) not treated independently.'

This project is *high* visibility, and the public perceives that the operation we are conducting is countering a big problem in society.

16. Choose 1 for ‘High visibility.’

Our external site needs to be mobile.

17. Choose 3 for ‘Mobile.’

Our remote site needs high visibility.

18. Choose 1 for ‘The REMOTE site is: high visibility.’

The main facility is located in the US.

19. Choose 1 for ‘Yes -- facility in continental US location.’

We are going to choose an existing site.

20. Choose 1 for ‘Yes -- an existing facility will be used as primary site.’

Since we have part of the system that is mobile, and they can/tend to be prone to natural interference (lightning, etc.).

21. Choose 2 for ‘Natural hazards are of some concern.’

We are somewhat concerned about services and their proximity.

22. Choose 2 for ‘Proximity for support services are of medium concern.’

Few sites that are of mixed security can guarantee that the system will be guarded.

23. Choose 1 for ‘Yes -- the system will have unattended time intervals.’

We can not afford redundant storage.

24. Choose 2 for ‘No -- redundant storage capabilities are NOT implied.’

SYSHIGH site is expensive and not in the budget, and most sites require contractors who are not SYSHIGH.

25. Choose 2 for ‘Personnel are mixed SYSHIGH and SYSLOW.’

Limiting personnel access to sensitive information is required.

26. Choose 1 for ‘Yes -- sensitive information access is a concern’.

Yes, we need a secure communication link to our remote/mobile site.

27. Choose 1 for ‘Yes -- secure communications links ARE needed.’

Yes, we may need to pass info on to other government agencies about our mission if needed.

28. Choose 1 for ‘Yes -- links to other systems ARE needed.’

No special hardware aspects are needed.

29. Choose 2 for ‘No -- special communications hardware is NOT needed.’

No special software is needed.

30. Choose 2 for ‘No -- special communications software is NOT needed.’

No, crypto needs are not required.

31. Choose 2 for ‘No -- crypto needs do NOT exist for communications.’

We are concerned about emanations/signals coming from our building.

32. Choose 1 for ‘Yes -- classified signals/emanations ARE of concern.’

We do not have an emanations security officer.

33. Choose 2 for ‘No -- NO emanations security officer is present.’

No, TEMPEST countermeasures are not necessary.

34. Choose 2 for ‘No -- TEMPEST countermeasures are NOT necessary.’

    Note: Answers to #32 and #34 suggest an inconsistency. This would be detected by an extension suggested in Section 5.1.

Since outside sites need classified data, we SHOULD separate/compartmentalize data.

35. Choose 1 for ‘Yes -- separation is necessary.’

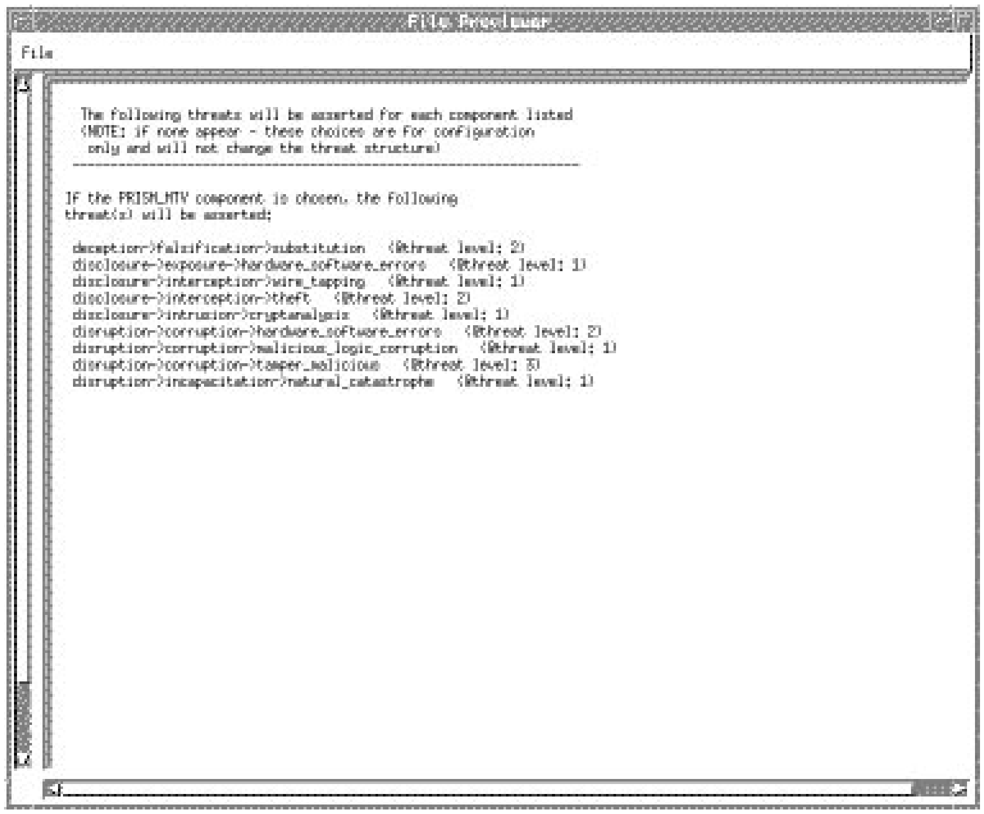
    Note: The Main Menu will reappear.

36. Choose 3 ‘Choose a particular configuration of the current component.’

37. Press Return to acknowledge.

Now we must choose/instantiate a particular configuration of the message processing system we’ve decided to reuse. Many configurations are possible, and each has a unique set of threats given to it.

For example, if we choose ‘help,’ we can see which particular threats are associated with the choices available.   Figure G-2 shows an example threats list for this demo.



```
 The following threats will be asserted for each component listed
 (NOTE: if none appear - these choices are for configuration
  only and will not change the threat structure)
-----------------------------------------------------------------

If the PRISM_MTV component is chosen, the following
threat(s) will be asserted;

 deception->falsification->substitution   (@threat level: 2)
 disclosure->exposure->hardware_software_errors   (@threat level: 1)
 disclosure->interception->wire_tapping   (@threat level: 1)
 disclosure->interception->theft   (@threat level: 2)
 disclosure->intrusion->cryptanalysis   (@threat level: 1)
 disruption->corruption->hardware_software_errors   (@threat level: 2)
 disruption->corruption->malicious_logic_corruption   (@threat level: 1)
 disruption->corruption->tamper_malicious   (@threat level: 3)
 disruption->incapacitation->natural_catastrophe   (@threat level: 1)
```

Figure G-2: Sample Threats List

Choose ‘None’ for now. We may not want to include these threats since they are optional.

38. Choose 2 for ‘None.’

The message processing system can have 1 of 2 types of “message generators.” ‘BB’ means bit-based. Again, we could examine the threats on each and determine which one would best suit our needs. For this demo, let’s select ‘ASCII_PRISM_msg_gen.’

39. Choose 1 for ‘ASCII_PRISM_msg_gen.’

The windowing system may not be of importance to us, so we can choose whichever we want. Choose mwm for this demo.

40. Choose 1 for ‘mwm.’

The version of X-windows, however, can make a BIG difference of threats. Choose X11_R5, since it is more up to date the R4.

41. Choose 2 for ‘X11_R5.’

This screen asks us which kind of ‘messages’ we want our system to process. Since we are concerned about DEA issues; choose 1, 2, and 4.

42. Choose 1, 2, and 4 for
    1 DEA_interdiction_ok
    2 inform_of_mechanism
    4 suspect_intercept

And finally, we do not need Interprocess Communication for our system, so choose None.

43. Choose 5 for ‘None.’

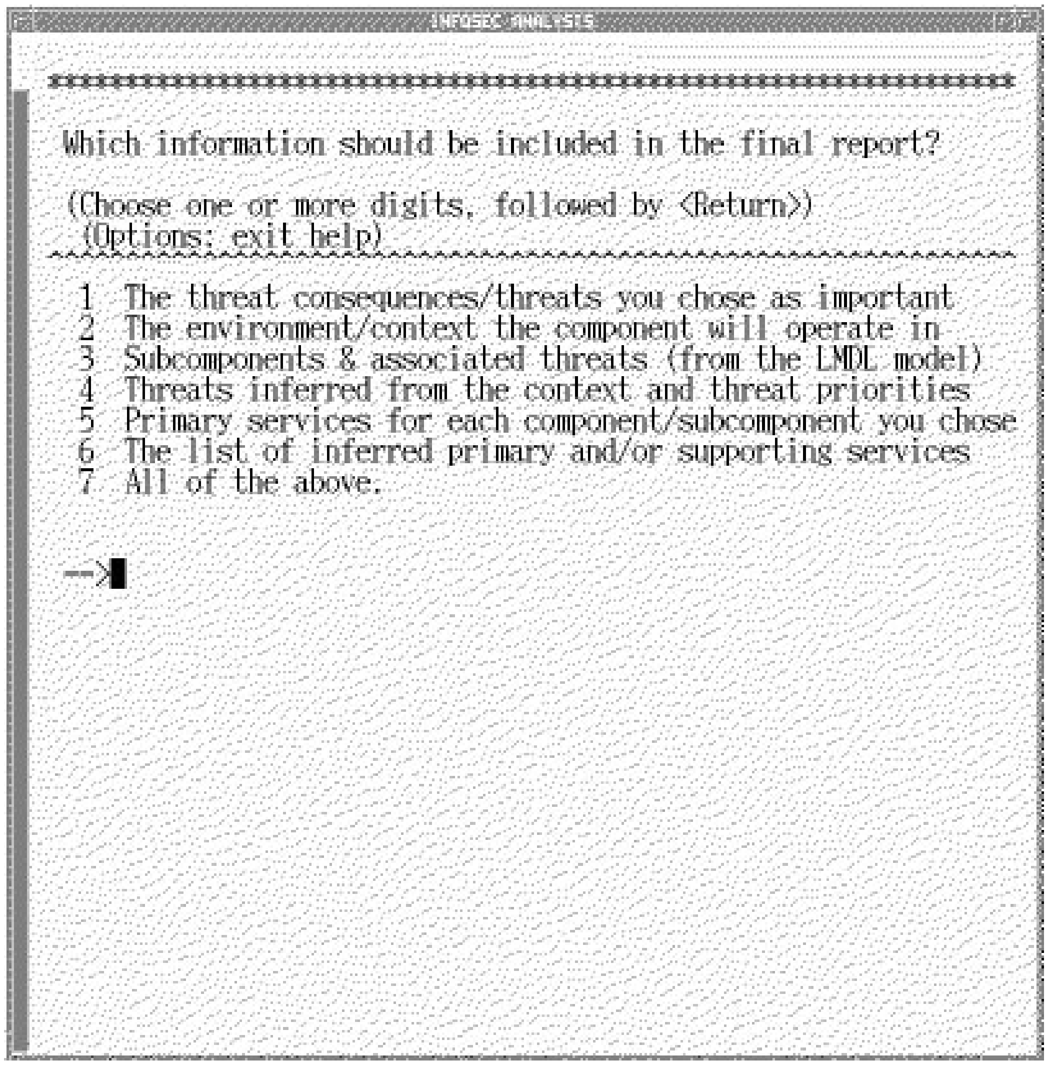
44. Press Return at “END OF COMPONENT TRAVERSAL” screen.

    Note: The Main Menu will reappear.

We are now at a point where we can get a report about the threats to our configuration of the message processor we want to reuse.

45. Choose 4 for ‘Create report and exit system.’

Note: The report configuration screen will now appear (Figure G-3).



```
***************************************************************

Which information should be included in the final report?

 (Choose one or more digits, followed by <Return>)
  (Options: exit help)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  1  The threat consequences/threats you chose as important
  2  The environment/context the component will operate in
  3  Subcomponents & associated threats (from the LMDL model)
  4  Threats inferred from the context and threat priorities
  5  Primary services for each component/subcomponent you chose
  6  The list of inferred primary and/or supporting services
  7  All of the above.


-->
```

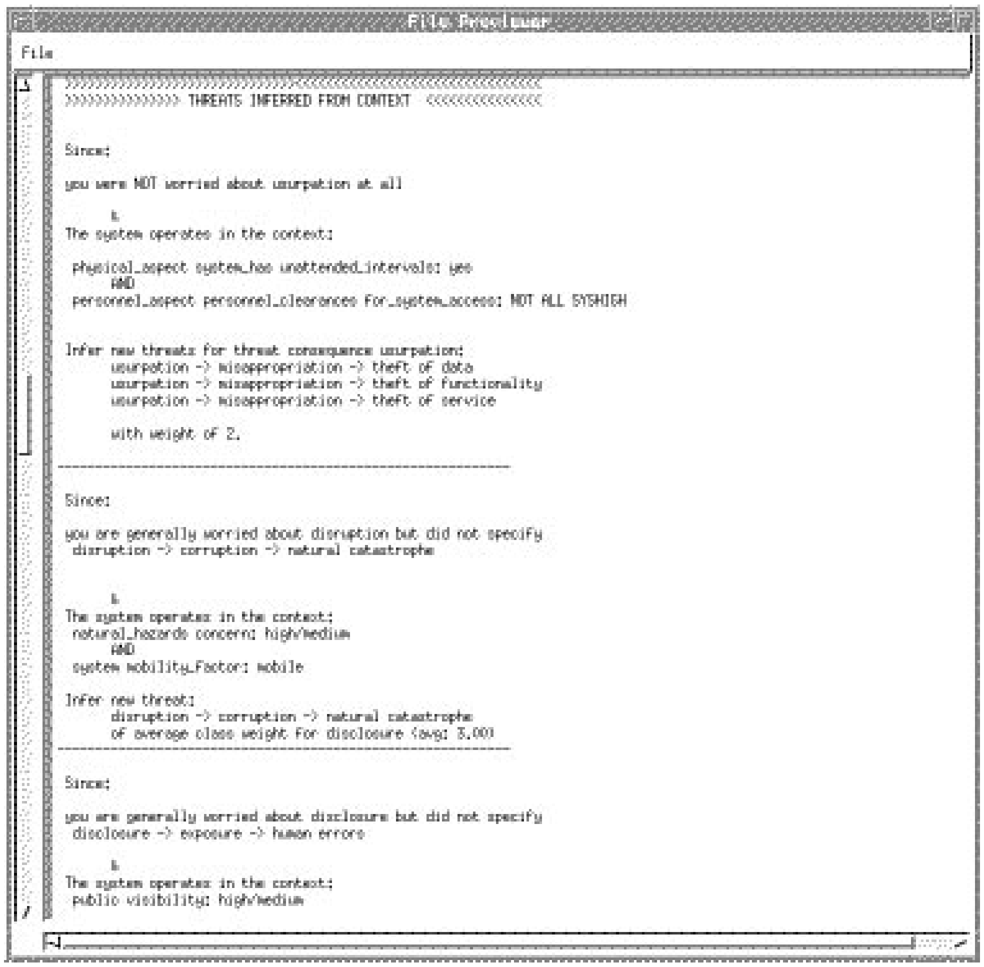
Figure G-3: Report Configuration Screen

We would like to see *everything* in our report.

46. Choose 7 for ‘All of the above.’

47. Press [Return].

Note: The report screen will now appear (see Figure G-4).



```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>> THREATS INFERRED FROM CONTEXT  <<<<<<<<<<<<<<<


Since:

you were NOT worried about usurpation at all

      &
The system operates in the context:

 physical_aspect system_has unattended_intervals: yes
      AND
 personnel_aspect personnel_clearances for_system_access: NOT ALL SYSHIGH


Infer new threats for threat consequence usurpation:
      usurpation -> misappropriation -> theft of data
      usurpation -> misappropriation -> theft of functionality
      usurpation -> misappropriation -> theft of service

      with weight of 2.
------------------------------------------------------------------

Since:

you are generally worried about disruption but did not specify
 disruption -> corruption -> natural catastrophe


      &
The system operates in the context:
 natural_hazards concern: high/medium
      AND
 system mobility_Factor: mobile

Infer new threat:
      disruption -> corruption -> natural catastrophe
      of average class weight for disclosure (avg: 3.00)
------------------------------------------------------------------

Since:

you are generally worried about disclosure but did not specify
 disclosure -> exposure -> human errors

      &
The system operates in the context:
 public visibility: high/medium
```

Figure G-4: Threats Report Screen

Look at the report. The first parts are simply reminders of which choices we made during the demo, i.e., on threats, components, and environment/context.

The last 3 sections, starting with ‘THREATS INFERRED FROM CONTEXT’ are of most importance. This section shows us *new*, additional threats that the user *should* be worried about, given the context of operation; the general threat worries stated; and the component involved. Reasons are given for each new inference.

If the user has chosen a threat consequence as important (along with 1 or more specific threats in that group), then an average value for that consequence group is applied as the ‘threat value’ for the inferred threat.

Below this section, labelled ‘Lists of primary services for each component you chose’ is the list(s) of primary services to counter each threat found for the components we have chosen in the message processing system.

Finally, the last section, ‘The following new primary and/or supporting service(s)’ shows any new primary and/or supporting services inferred to help form a security lattice network according to context. In our example, only 1 additional primary service was found. Other paths in this demo may show several, or none of each kind, depending on choices made.

Security architects can use this report to significantly bolster any security holes in the systems they design. The report offers a window into the threats/services available to each reusable component, therefore speeding development and reducing costs, etc.

Appendix H Domain Specific Software Architectures (DSSA)

Domain specific software architectures have arisen out of a desire to reuse previous design decisions, and the observation that very few development efforts are involved in completely novel applications or approaches. Applications that used to be novel (e.g., avionics systems) are now becoming quite commonplace, and are ripe for technological solutions to decrease the risk associated with building them. No production development would construct a compiler today from scratch without regard to the structure, components, or solution strategies used in other modern compilers. Nobody would build such a compiler without employing a parser generator at the very least, and perhaps optimizer generators as well. Similarly, in a domain where once-novel applications have been fielded often enough, a body of engineering knowledge has arisen about them. Part of that corpus is the software architecture of a set of successful members of that application family. Domain specific software architectures are, fundamentally, an exercise in knowledge reuse.

Technology to support domain specific software architectures occurs along a spectrum, sketched in Figure H-1. At the lowest productivity end of the spectrum, designers create the architecture for each new system on an *ad hoc* basis, without regard for lessons learned from previous, similar development efforts. Above this, in the reuse-without-automation realm, domain analysts have assembled architectures that represent many of the members of the domain; the reusable assets are documents that describe or represent those architectures. A designer of a new system consults this “library” of architectures and takes advantage of the design decisions, represented by those architectures, that were resolved in the building of previous systems in the family.

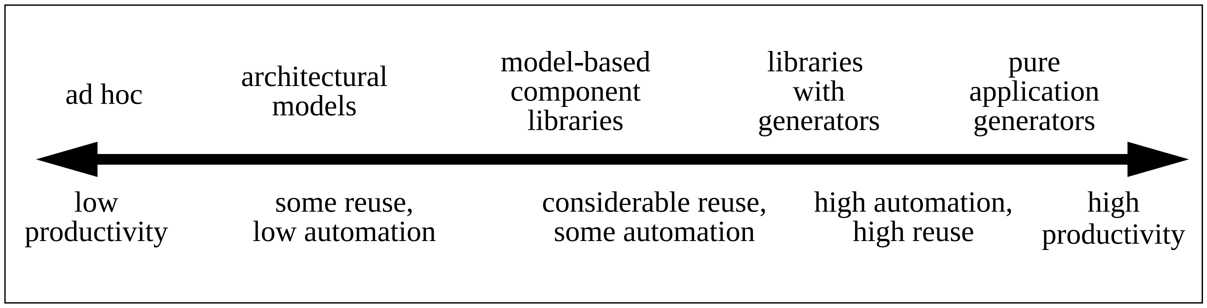

ad hoc
architectural models
model-based component libraries
libraries with generators
pure application generators
low productivity
some reuse, low automation
considerable reuse, some automation
high automation, high reuse
high productivity

Figure H-1: A Spectrum of Architecture Selection Technology

Somewhat farther up the spectrum occur architectures whose components are instantiated in the library. A new designer can take advantage of the architecture, plus import actual copies of the components (typically, modules in the form of source code). Ideally, the components are well-suited for the new application; typically, however, they require modification. The necessary modification to the checked-out components may be made by hand, or (in a more sophisticated library environment) the components may be parameterized. Modifications can then be represented by instantiating the parameters.

Application generators represent the high end of the technology spectrum. These are systems that allow a user[1] to specify all relevant characteristics of the desired application. The architecture for the application family, as well as the components (or the ability to create the components) reside in the generator. The generator instantiates the components that, taken together, satisfy the requirements imposed by the application engineer. Figure H-2 illustrates the process from the application engineer to the developed application. Examples of this technology include the GenVoca-based [Batory 94a] application generators such as Genesis [Batory 86, Batory 94b].

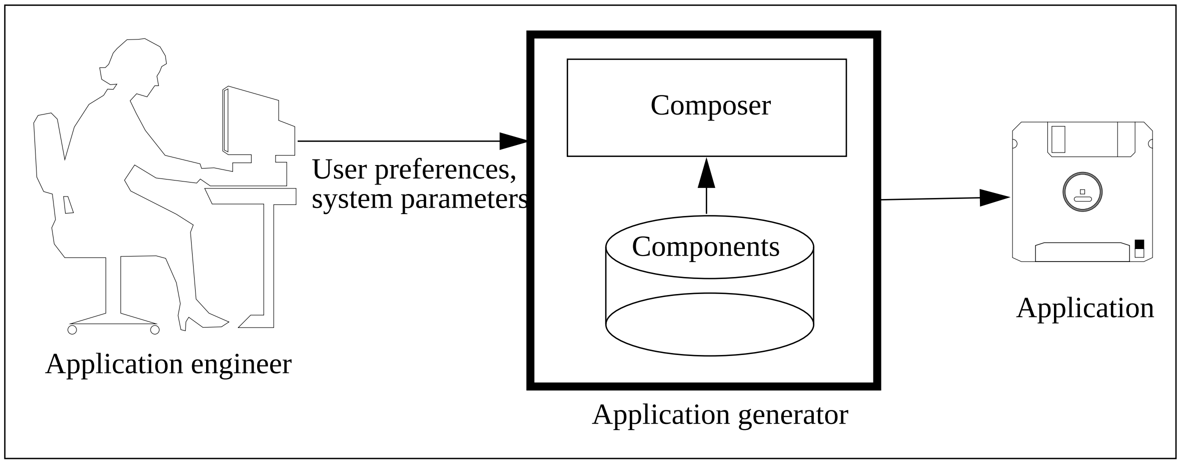


Figure H-2: An Application Generator

1. To avoid confusion with the end user of the application, we will call the builder of the system (or the user of the application generator) an *application engineer*.

References

[Batory 86]	Batory, D.; Barnett, J.; Garza, J.; Smith, K.; Tsukuda, K.; Twichell, B.; & Wise, T.*Genesis: A Reconfigurable Database Management System* (TR-86-07). Austin, TX: University of Texas at Austin, 1986.
[Batory 94a]	Batory, D.; Singhal, V.; Thomas, J.; Dasari, S.; Geraci, B.; & Sirkin, M. “The GenVoca Model of Software-System Generators.” *IEEE Software* 11,5 (September 1994): 89-94.
[Batory 94b]	Batory, D.; Thomas, J.; Sirkin, M. “Reengineering a Complex Application Using a Scalable Data Structure Compiler,” 111-120. *Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*. New Orleans, LA, December 6-9, 1994. New York, NY: ACM Press, 1994.
[CARDS 94]	Central Archive for Reusable Defense Software. *Technical Concepts Document* (STARS-VC-B009/001/00). January 1994.
[Cohen 92]	Cohen, Sholom G.; Stanley Jr., Jay L.; Peterson, A. Spencer; & Krut Jr., Robert W. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain* (CMU/SEI-91-TR-28, ADA256590). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.
[Giarratano 93]	Giarratano, Joseph C. *CLIPS User’s Guide, Version 6.0*. Clear Lake, TX: NASA, Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, 1993.
[Gulachenski 94]	Gulachenski, B. & Costa, M. *Taxonomy of Threats and Security Services for Information Systems* (Working paper: Project No.:8353Z, Contract No.:DAAB07-94-C-H601). Bedford, MA: Mitre Corporation, January 18, 1994.

[Kang 90]	Kang, K.; Cohen, S.; Hess, J.; Novak, R.; & Peterson, S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University,1990.
[Krut 94]	Krut Jr., Robert W. *Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology* (CMU/SEI-93-TR-11). Pittsburgh, PA:Software Engineering Institute, Carnegie Mellon University,1993.
[NASA 93a]	National Aeronautics and Space Administration. *CLIPS Reference Manual Volume I Basic Programming Guide, CLIPS Version 6.0*. Clear Lake, TX: Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, June 2, 1993.
[NASA 93b]	National Aeronautics and Space Administration. *CLIPS Reference Manual Volume II Advanced Programming Guide, CLIPS Version 6.0*. Clear Lake, TX: Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, June 2, 1993.
[NASA 93c]	National Aeronautics and Space Administration. *CLIPS Reference Manual Volume III Interfaces Guide, CLIPS Version 6.0*. Clear Lake, TX: Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, June 2, 1993.
[Rumbaugh 91]	Rumbaugh, James, et al. *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.
[STARS 92]	Software Technology for Adaptable, Reliable Systems*. RLF Graphical Browser User's Manual* (STARS-TC-04046/005/00). July 1,1992.
[STARS 93a]	Software Technology for Adaptable, Reliable Systems*. RLF Modeler Tutorial* (STARS-UC-05156/020/00). February 1993.
[STARS 93b]	Software Technology for Adaptable, Reliable Systems*. RLF User's Manual, Version 4.1* (STARS-UC-05156/013/00). March 1993.

[Wallnau 88] Wallnau, K. “Construction of Knowlegde-Based Components and Application in Ada,” 3/1-21. *Proceedings of AIDA-88, Fourth Annual Conference on Arificail Intelligence and Ada*. Fairfax, VA, Nov. 15-16, 1988. Fairfax, VA: George Mason University, 1988.