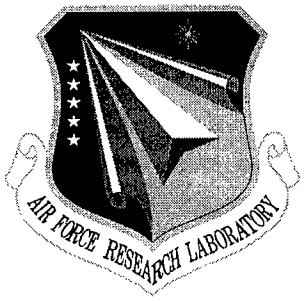
AFRL-IF-RS-TR-2001-238
Final Technical Report
November 2001



DESIGNEXPERT

CoGenTex, Inc.

Lee Ehrhart, Tatiana Korelsky, S. Daryl McCullough, Joseph McEnerney, Benoit Lavoie, Scott Overmyer, Owen Rambow, and Franklin Webber

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

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

AFRL-IF-RS-TR-2001-238 has been reviewed and is approved for publication.

APPROVED: Robert M Flo

ROBERT M. FLO
Project Engineer

FOR THE DIRECTOR: James W Cusack

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

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

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

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	NOVEMBER 2001	Final Jun 96 - Oct 99

4. TITLE AND SUBTITLE
DESIGNEXPERT

5. FUNDING NUMBERS
C - F30602-96-C-0076
PE - 62702F
PR - 5581
TA - 32
WU - 13

6. AUTHOR(S)
Lee Ehrhart, Tatiana Korelsky, S. Daryl McCullough, Joseph McEnerney, Benoit Lavoie, Scott Overmyer, Owen Rambow, and Franklin Webber

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CoGen Tex, Inc.
840 Hanshaw Road
Ithaca New York 14850

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFSB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2001-238

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Robert M. Flo/IFSB/(315) 330-2334

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Many failures of software systems may be traced to inadequate consideration of systemwide requirements such as system security, reliability, and usability. None the less, commercially available computer-aided software engineering tools do not provide any special assistance for satisfying them, instead concentrating on a system's functionality. This report presents DesignExpert, a knowledge-based tool intended to aid system developers and other stakeholders to effectively address system-wide requirements. The tool elicits requirements, analyzes needs, generates design alternative, and suggest evaluation strategies.

14. SUBJECT TERMS
Computer-Aided Software Engineering, Computer Security, Fault-Tolerance, Human-Computer Interaction

15. NUMBER OF PAGES
64

16. PRICE CODE

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

Contents

List of Figures

Chapter 1

Introduction

Many failures of software systems can be traced to inadequate analysis of system-wide requirements such as reliability, security, and usability. DesignExpert is a software tool that helps its user to address such system-wide requirements effectively. In the DesignExpert approach, the user is guided to state the system-wide requirements clearly and is also made aware of design alternatives for satisfying the stated requirements. Using a variety of knowledge-based techniques, DesignExpert helps a user who is not an expert in reliability or security techniques to evaluate alternative requirements and designs. DesignExpert used in conjunction with commercially available tools for building software functionality will reduce the risk that reliability or security requirements might be neglected during the design process.

DesignExpert consists of three components:

1. a Security Assistant (SA);
2. a Fault Tolerance Assistant (FTA);
3. a Human-Computer Interaction Assistant (HCIA).

This report describes the overall system and the shared vision that underlies its three components in Chapter 2. The three components, the FTA, SA, and HCIA, are described in more detail in Chapters 3, 4, and 6, respectively. In Chapter 5, we discuss in more detail the explanation facility for the reasoning of the SA.

Chapter 2

DesignExpert

2.1 Introduction

After more than three decades of experience and tremendous advances in hardware and software technologies, organizations with critical system requirements are finding that the systems delivered still fail to meet the operational need. In many cases, the problem lies in a failure to correctly identify and address the system-wide requirements (such as usability, security, fault-tolerance, maintainability, and real-time requirements) that define the quality and effectiveness of system performance. Commercially available computer-aided software engineering (CASE) tools do not provide any support for identifying and meeting system-wide requirements, concentrating exclusively on developing and rapidly prototyping application domain functionality. A considerable body of knowledge about system-wide properties has accumulated and can be made available to developers.

This report presents DesignExpert, a computer-aided software engineering (CASE) tool intended to aid system developers and other stakeholders to effectively deal with system-wide requirements.

This overview over the system presents an overview of system-wide requirements and the related life cycle development issues in Section 2.2. Section 2.3 follows with an overview of DesignExpert, and Sections 2.4, 2.5, and 2.6, respectively, discuss in detail the three domains of expertise it supports: human-computer interaction, security, and fault tolerance. The paper concludes with a brief discussion of a formal evaluation effort.

2.2 System-Wide Requirements

2.2.1 Importance of System-Wide Requirements

System-wide requirements, often called nonfunctional requirements, define and constrain most or all of the functional modules of a system rather than a small subset (Davis, 1994). Common examples of system-wide requirements are the performance, usability, or maintainability of a system. DesignExpert concentrates on three particular system-wide requirements:

security, fault tolerance, and human-computer interaction. Each of these requirements is both non-functional and system-wide:

- A system is secure if it protects data in specific ways, from unauthorized disclosure or tampering, or from unavailability. Security differs from functionality because data protection may be needed regardless of how the system is expected to process its data. Security can sometimes be enforced in a single module, a gateway to the system, but more typically it depends on the interaction of many modules that form the trusted part of the system. A mistake in implementing one of these modules can easily compromise the security of the entire system.

- Similarly, a system is considered fault tolerant if it will continue to function correctly in spite of one or more failures of its components. Fault tolerance should not change a system's functionality much, if at all, but rather should protect that functionality from specific kinds of failures. Because failures can occur in any module, fault tolerance is necessarily a system-wide concern.

- Finally, human-computer interaction design involves the allocation of tasks between operator and system, presentation of information, and the ergonomics of interaction. Thus, human-computer interaction involves more than the functionality of the system's external interfaces — it involves the overall system concept.

Thus, each of these requirements depends on most or all of the system; none could be satisfied simply by adding or fixing a single module unless most or all of the other modules were already designed with the system-wide requirement in mind. Any new module added to the system would be affected by these system-wide requirements. As a result, while system-wide requirements are important throughout the software engineering process, they are particularly important at the early stages: requirements gathering and formulation, and design.

2.2.2 Communicating about System-Wide Requirements

System engineering is a complex cognitive task involving many stakeholders with specialized knowledge, needs, and expectations. As in all cooperative human problem-solving tasks, communication among stakeholders plays a crucial role in successful system engineering. If automated tools are used, the need for successful communication extends to the interaction between human stakeholders and the automated tools. Communication is particularly important during the requirements gathering and analysis phase, since this phase involves the elicitation and interpretation of informally expressed domain knowledge, and it involves a great number of stakeholders of different technical background (clients, domain experts, requirements engineers, system engineers, end users). The requirements engineer must obtain information about the domain and requirements from domain experts and clients . The requirements engineer must formulate these requirements and validate them with the end user or client, who is typically not versed in certain software engineering methodologies and

notations. However, the effort of pinpointing the exact requirements is crucial if subsequent system development is to avoid the costs associated with past software disasters or retrofitting.

In addition, since system-wide requirements can imply a wide array of very specialized knowledge, we must assume that most of the stakeholders involved in systems engineering are not experts in all the relevant types of system-wide requirements. Lack of expertise makes communication even more difficult, and even more crucial: for example, a client will need to know at all stages of the requirements engineering process why certain security requirements have been posited by the requirements engineer, and he or she needs to do be able to have the requirements explained in a manner that is comprehensible to him or her.

Finally, the standardization of documentation procedures (such as those defined by DoD STD 2167A or IEEE Std 1498) places considerable burden on the participants in the system engineering effort, especially in those domains in which they are not experts.

2.2.3 The Need for Support in the Development of System-Wide Requirements

While system functionality typically can be changed by modifying a few modules, the non-functional requirements constrain how functionality is delivered by the system. Because a system-wide requirement affects many or all modules in a system, "retrofitting" the system to satisfy the requirement usually has a considerable cost. There is a pressing need for low-cost design, development, and testing environments that permit assessments about which investments in system-wide requirements make sense and which should be avoided. Support for these assessments must be available in an understandable form to the persons making acquisition decisions for an organization and shared with the teams responsible for developing and maintaining the system.

The integral nature of system-wide requirements demands their consideration across the development life cycle to ensure delivery of expected functionality. Developers need economical and effective tools and methods for defining system-wide requirements, designing the requisite qualities into the functional solutions, and evaluating software products (e.g., designs, prototypes, and completed systems) with respect to the system-wide requirements.

Specifically, we have identified the following needs:

- Requirements engineers who are not specialists in a particular type of system-wide requirement may not know what information to elicit from clients and domain experts.
- Requirements engineers may have difficulty in drafting relevant documents because they do not know what needs to be recorded, nor what language is appropriate.
- Clients (such as Government program managers), requirements engineers, and designers may have problems estimating what requirements can be realistically met within the given constraints of time and budget.
- System designers may not know how to address certain system-wide requirements.

However, despite the recognized importance of system-wide requirements, commercially available computer-aided software engineering (CASE) tools do not provide any special assistance for satisfying them, instead concentrating on support for developing and rapidly prototyping a system's functionality. Meanwhile, a considerable body of knowledge about system-wide requirements has been accumulated and can be made available to developers. The goal of DesignExpert is to make this knowledge available to different stakeholders in convenient manner.

2.3 System Overview

DesignExpert is a knowledge-based CASE tool composed of three assistants, each giving advice in its own domain of specialization:

- The Human-Computer Interaction Assistant (HCIA) provides advice on the design and evaluation of information presentation and interaction protocols.
- The Fault-Tolerance Assistant (FTA) provides advice on employing hardware and software resources to achieve a given level of fault-tolerance.
- The Security Assistant (SA) provides advice on issues relating to data security.

The three assistants are conceptually independent of each other. This means that the user could use a single one of the three assistants; there is no need to use all three assistants at each session or for each project. At the same time, they share basic concepts of interaction. Each elicits information from the analyst by requesting answers to specific questions about the system to be designed. Each provides a recommendation and/or analysis of the data.

DesignExpert supports the software practitioner who is not an expert in the particular domain of system-wide requirements and allows him or her to use this knowledge from the earliest phase of requirements analysis through testing and validation of the resulting system. Specifically, the tools help the analyst:

- define organizational needs and the corresponding system-wide requirements;
- formulate required documents and documentation;
- evaluate requirements in terms of constraints of project time line and budget; and
- evaluate designs for conformance to identified requirements.

The analyst can explore the recommendations and/or analyses given by one of the assistants by changing the system information, and by obtaining explanations in English of the recommendation. The system can also generate reports (in English) summarizing the information about the system and/or the recommendations.

DesignExpert addresses all phases of systems engineering, but concentrates on the requirements analysis and design phases. The tool itself does not presuppose any particular software

engineering process and can be used in the context of any processes, as long as the process has steps corresponding to the standard phases. In addition, the flexible report generation facility can easily be adapted to help satisfy the reporting and documentation needs of any specific process.

To increase its utility, DesignExpert is implemented using a platform-independent architecture. The advisory function is implemented in CLIPS 6.0. The user interface is implemented in HTML using CGI and JAVA to handle input/output and serve sessions. This interface architecture permits economical use of multimedia on UNIX, Windows, and Macintosh platforms.

2.4 The Security Assistant: Supporting System Integrity

The Security Assistant (SA) supports the formulation of security requirements and choice of security mechanisms to meet those requirements.[1] In this process, SA helps the user balance the cost of countermeasures and the likelihood that a threat will be actualized, on the one hand, with the value of assets and the importance of the system mission, on the other.

A security requirement is a written formulation of the objective of opposing some attack on, threat to or vulnerability of the given system. System security requirements derive from several sources, among which the most noteworthy are:

- Mission needs and concepts of operations
- Risk and threat assessments of potential deployment sites
- Requirements stipulated for similar systems
- Organizational security policies
- Mandatory requirements (for government systems)

The SA uses a series of HTML surveys (questionnaires) to gather security-related information about a system's functional architecture, and its deployment environment into the SA repository and analyze that data for consistency and completeness.

The SA produces two kinds of output:

- The SA automatically generates reports in fluent English that summarize the information entered into the repository.
- Given the information about the system functionality and its deployment environment, the SA provides a critique of chosen security measures and recommends countermeasures of appropriate strength to offset known or potential risks. This critique is also presented to the user in the form of a report automatically generated in fluent English.

[1] For a more complete exposition of the theoretical approach underlying the SA and the FTA, see (Webber et al., 1998).

The Security Assistant is designed to provide support to users during the pre-implementation phases for new systems and maintenance and upgrade phases for legacy systems. Several different groups of users might benefit from the SA's advice:

1. Requirements engineers can use SA to maintain a representation of the intended system's security requirements as the system's requirements evolve (and, later, as the system itself evolves). They can use SA to ascertain that they have gathered the relevant information. The report generation capability facilitates the communication about requirements with other stakeholders and the documentation of the requirements.

2. System designers benefit by including security as a system wide property entertwined with functional and architectural design and not as an add on. Moreover, this group benefits because SA can handle generic security concerns thus giving them the freedom to concentrate on the really unique security problems that are specific to each system.

3. Government system certifiers, who are responsible for determining the residual risk associated with deploying these systems for a given mission in a given environment often have to treat each case separately. Moreover, they must contend with a mountain of required paper work as well as perform technical analyses. This group benefits by having a DITSCAP (DoD Information Technology Security Certification and Evaluation Process) compatible framework in which to evaluate a system. A future version of SA is planned that will provide comprehensive support DITSCAP's System Security Authorization Agreement (SSAA) as a living document and will provide certifiers with the means to assign ITSEC classes to systems and make comparisons.

4. System Administrators and IT Managers benefit by having a tool that can help organize and analyze systems and networks that they are responsible for maintaining. These professional are frequently tasked with enhancing such systems by deploying new applications or hosts. This can cause security compromises for either new or old components. These decision makers, who have to bear the responsibility for the choices they make, need objective and cost conscious advice that reflects the IT system mission as well as the risks to which it is subjected. Currently, decision makers rely solely on the advice of domain experts.

2.4.1 SA's methodology of system description

Modern distrubuted systems applications typically consist of hardware and software components that are themselves systems. This leads to the notion of "system of systems". From a security standpoint this is a very important perspective since not only must component systems be trustworthy but also they most be combined in such a way that the resulting system of systems is also trustworthy. The Security Assistant has been designed with this perspective in mind.

A system under study is described as a set of interrelated functional components which are deployed (and possibly replicated) at sites. Viewed from the topmost level the target system has a global operational environment which is made up of the site environments. Threats

to security come from these enviroments and can arise from natural or human sources. It is this threat space that characterizes the risks run by the target system. Ultimately, security of the target system is achieved by determining and implementing a Security Policy which counters the threats and enables the system missions to serve its clients while protecting its assets. Moreover, the importance of missions and the value of assets must be weighed against the cost of countermeasures to the threats in the environment.

Although a target system security policy is the ultimate goal, elements of it are often known in advance or are dictated by organization policy. For these reasons, several security policy elements appear as both inputs and outputs of the SA.

In SA's methodology, the security-related information about the system is gathered at four levels:

1. The system-wide level. This survey gathers information about the system's functional architecture, its missions and clients, its main assets, its system-wide security charachteristics, and the list of sites that constitute its deployment environment.

2. The site level. A site is seen by SA as a collection of hosts that are usually concentrated geographically in one region, logically related by common purpose, physically connected to each other by a LAN, pre-exist the target system, and often perform functions unrelated to the target system. The site survey gathers information about what system assets are deployed at the site, detailed information about the site's physical, computing and communications environment, as well as its detailed security characteristics.

3. The host level. Hosts are seen by SA as connected to the site to provide the platforms onto which target system functional components are deployed. Hosts are described by specifying hardware configuration, operating system, the set of deployed target system software components which implement system assets described on the higher levels, and the suite of other installed applications (if any) and libraries.

4. The software component level. The software component level provides the user with a place to characterize how host resources are shared between the target system software and peer software on the same host. The more detailed information about the peer applications running on each host is available, the more security vulnerabilties can be detected and countered during the requirements phase.

2.4.2 Automatic reasoning for security analysis and advice

SA uses an expert system to analyze and critique user descriptions of a system under study. The current SA prototype is targeted primarily at the requirements analysis phase of system development and not at the maintenance phase or the reengineering of the security characteristics of the legacy systems (the SA's framework nevertheless is general enough to accomodate these latter tasks as well). Consequently, the reasoning engine of the current uses information of the two higher levels (system-wide and site level) but not of the two lower levels (host and software component) which are rarely specified in detail at the requiremetns phase.

Knowledge of security derived from expert experience, publications and DoD standards such as the TCSEC and the Common Criteria have been encoded in rules that are activated by the presence or absence of relevant facts in the SA's repository.

By determining what types of damage an asset or mission can sustain, what kind of attack methods are possible, how likely it is that an attack can be mounted and what system defenses are available to counter attack methods it is possible to formulate inference engine rules that reflect rules of thumb like the following:

- Make sure that every asset is protected from every known potential attack method.
- Balance the cost of the system defenses with the asset values and the mission importance.

2.5 The Fault Tolerance Assistant: Supporting System Reliability

The Fault Tolerance Assistant (FTA) is DesignExpert's support for developing reliable distributed computer systems. Using FTA helps to posit realistic reliability and availability requirements, and increases confidence that a system's design satisfies its reliability and availability requirements.

FTA surveys its user's requirements for system reliability and availability. A *reliability* requirement tells how likely it is for a system to give continuously correct operation for a given duration. An *availability* requirement tells how likely it is for a system to be operating correctly at any given time. Several similar requirements can also be specified in FTA (Siewiorek and Swarz, 1982):

- the mission time;
- the mean time to failure (MTTF);
- the mean time between failures (MTBF);
- the mean time to repair (MTTR).

The availability, MTBF, and MTTR requirements depend on the possibility that failed components of the system can be repaired, while the others do not.

To analyze the system requirements, the FTA user must supply an abstract model of the system. This model is hierarchical: every component and subcomponent may be composed of other subcomponents. The system model includes both hardware and software.

A textual representation of the model is built in the Acme architecture description language(Garlan et al., 1997). We chose the Acme language to be the standard architecture description for FTA because it is a simple language designed to allow easy translation to

the architecture description languages used by other tools. The Acme language also can be easily extended by associating attributes with each component of an architecture.

The system model can include the following attributes for each component:

- replication factors;
- failure and repair rates;
- number of failures tolerated in each replicated ensemble;
- the failure model (e.g., Crash; Byzantine) assumed and the failure model guaranteed;
- the assignment of each software component to the hardware component on which it runs.

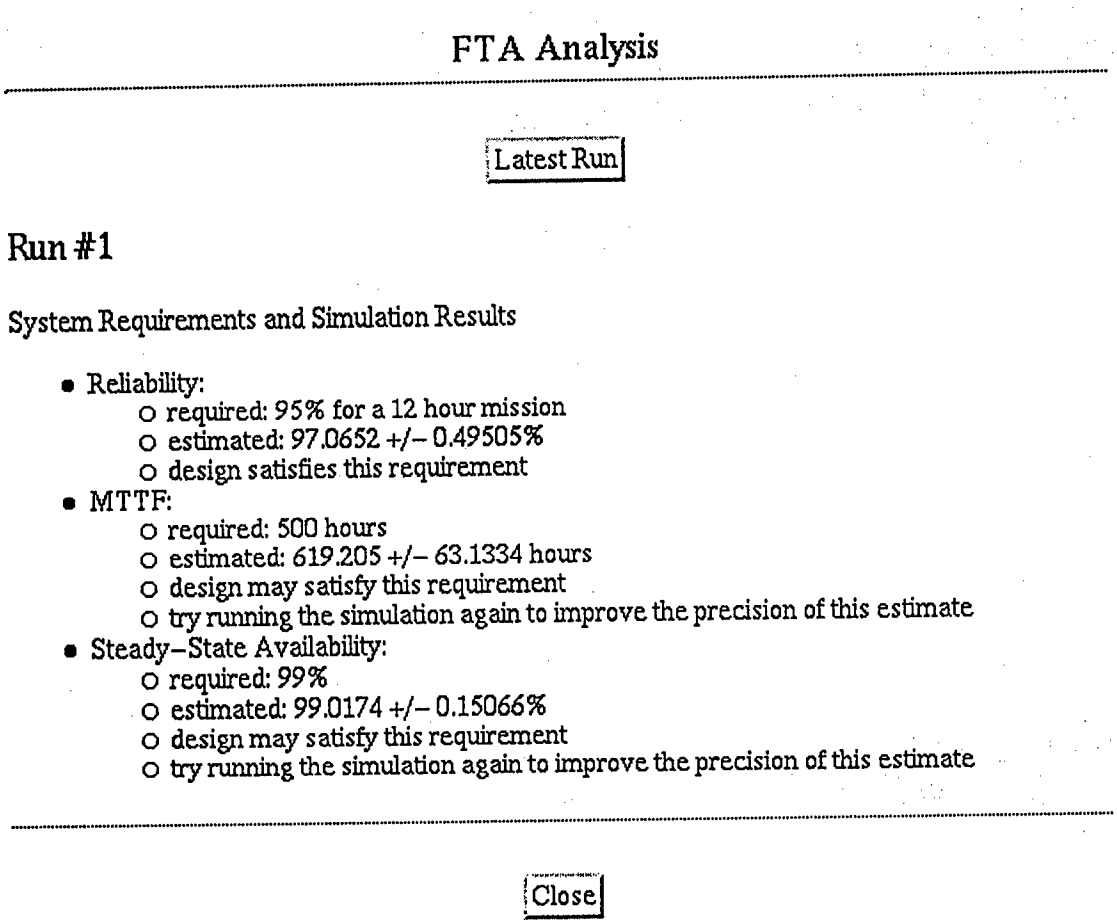


Figure 2.1: Sample FTA analysis

FTA produces these outputs:

1. A Monte Carlo simulation of the system model. This simulation assumes that each atomic element of the system model undergoes failure and repair at constant rates. The steps in the simulation can be made visual using a graphical architecture editor developed at Key Software.

2. Estimates of reliability, availability, and other measures of goodness of a given system model. To estimate reliability, mission time, and MTTF, the simulation is run many times and the results averaged. To estimate availability, MTBF, and MTTR, a very long simulation is run in which the system can fail and be repaired many times and averages are computed from that single run. FTA also estimates the uncertainty in each of these averages.

3. A critique of the attributes in the system model. FTA looks for several kinds of inconsistencies in these attributes, including: whether the fault tolerance specifications can be achieved given the model of failures; whether obvious failure correlations exist; whether the failure model assumed by a component X can be guaranteed by the components on which X depends.

4. Recommendations for increasing reliability. Improvements in reliability can sometimes be had by increasing the degree of replication and sometimes by changing the allocation of software components to processors(Nieuwenhuis, 1990). FTA estimates the potential for such improvements using a "quick and dirty" analysis rather than using the computationally-intensive (but more accurate) Monte Carlo simulation.

For a sample of an FTA analysis output, see Figure 2.1.

FTA's design advice should lead to better system models and ultimately to a sound judgment of whether particular reliability requirements can be satisfied. If a model satisfies the requirements, the number and kind of components in the model can be used to estimate the final cost of building the real system.

The users of the FTA include the following groups:

- Customers (such as Government program engineers) and requirements engineers can estimate whether requirements for fault-tolerance are realistic.

- System designers can ascertain that a system's design satisfies its reliability and availability requirements.

FTA's requirements survey can be reached from DesignExpert's Security Assistant (SA) (see section 2.4), thus coupling the two assistants together.

2.6 The Human-Computer Interaction Assistant (HCIA)

Human-computer interaction design embodies most of the system concept that is "available" to the user to guide his/her mental model of the system. For example, the HCI design incorporates such critical system design factors as:

- representation of information regarding the situational elements external to the system (support systems, environment, threats, etc.);
- representation of system states and feedback to the operator on results of actions taken;
- allocation of tasks between the human users and the computer as determined by the dynamics of the situation and the requirements of the methods selected to support task performance;
- and modes in which users may interact with all of this information to explore situations, develop hypotheses, generate options, select among alternatives, and implement their decisions.

The goals of the HCIA, include support for the following issues:

1. Cognitive Engineering: Design of displays and interaction routines consistent with what we know about human cognitive structures and processing; displays that are consistent with human data organization and "storage," and the way humans frame problems, make inferences, generate options, and implement action;
2. Organizational Focus: Permitting the user to examine HCI designs in the context of the larger system issues, such as its integration with other systems and coordination across users, teams and organizations;
3. Technology exploitation: Suggesting the innovative use of image, text and graphics processing technology to support real-time animation, user or system-directed searches, multidimensional displays, and "virtual" processing environments to improve the transparency of the interaction between the users and their tasks;
4. Designing for error: design of information displays and interaction protocols for preventing and controlling the impacts of operator error to improve system performance and reliability; and
5. Evaluation: Suggesting techniques for evaluating the proposed HCI design and new system concepts to identify potential risks not only to overall system performance, but also to the system development effort.

The HCIA has two components. The **HCIA Advisor** addresses all phases except evaluation. It was designed and implemented by Drexel University (under the supervision of Dr. Lee Erhart, following previous work under the name DesignPro) and later reimplemented and improved by CoGenTex. The **HCIA Critic** addresses the evaliuation phase. It was designed and implemented by CoGenTex with support from Dr. Scott Overmyer of Drexel University. The current version of HCIA Critic focuses on HTML-based interfaces.

2.7 Summary

The DesignExpert approach derives requirements from models of system and user needs at varying levels of abstraction to enhance understanding and consideration of system-wide requirements across the system development life cycle. In the earliest phases of development, a broad-brush approach may be used by planners and analysts to estimate feasibility and scope projects. The initial profiles and models can be elaborated during requirements analysis for greater specificity and used to generate and evaluate design alternatives. The report facilities extend the documentation capabilities and provide a summary of requirements and design issues for use by the implementation and testing teams.

The final prototypes of the three assistants have been implemented. DesignExpert will be formally evaluated in the immediate future (using both a focus-group evaluation and a use-based evaluation). We intend to report on the results of the formal evaluation in the final paper.

Chapter 3

The Fault Tolerance Assistant

3.1 Overview of FTA

The dependability of computer software and hardware has been a central concern of system designers since the first computers were built. Computer systems fail as the result of failures in both their hardware and software components. If a system depends on a single component then the failure of that one component can crash the entire system. In contrast, a fault tolerant design prevents certain kinds of component failures from causing system failure. Fault tolerance is achieved through adding redundancy, often redundant components, to a system.

As more components are added to a system the probability of system failure tends to increase. Thus, a distributed system with components running on many hardware elements will tend to have more ways to fail than a centralized system built for the same purpose. Fault tolerant engineering techniques use redundancy to counter this tendency by making system failure less dependent on individual components. These techniques have become essential for dependable design of large distributed systems.

The Fault Tolerance Assistant (FTA) is a tool to help with the development of dependable distributed computer systems. Using FTA helps make certain that a system's design, and indirectly its implementation in code, satisfies the system requirements for dependability.

3.2 FTA Support for Potential Users

The FTA offers several kinds of assistance to its user:

- FTA helps its user to state the dependability requirements for a specific system. The next section discusses the various kinds of requirement it supports.
- FTA estimates whether the requirements can be satisfied for a particular system model.
- FTA may recommend changes to the system model that improve dependability.

- FTA simulates the behavior of the system model as components fail and are repaired. The simulation can be viewed pictorially.

FTA's capabilities will be most useful to several different kinds of system stakeholders:

- System analysts will use FTA's requirements analysis to gain quick feedback on the feasibility of system-wide dependability requirements.
- System implementers will use FTA's system modeling facilities to build fault tolerance into the code according to a uniform plan for the entire system.
- Program planners will use FTA's requirements analysis to get a rough estimate of the eventual cost of a system.

While FTA may be useful at various stages of the software life cycle, its focus on requirements analysis makes it most useful in the early stages of software development.

3.3 Inputs to FTA

3.3.1 Requirements

FTA surveys its user's requirements for system reliability and availability. A *reliability* requirement tells how likely it is for a system to give continuously correct operation for a given duration, called the *mission time*. An *availability* requirement tells how likely it is for a system to be operating correctly at any given time. Several similar requirements can also be specified: the mean time to failure (MTTF); the mean time between failures (MTBF); the mean time to repair (MTTR). The availability, MTBF, and MTTR requirements depend on the possibility that failed components of the system can be repaired, while the reliability, mission time, and MTTR do not. Requirements of these kinds are important for any system that needs to tolerate the failures of some of its hardware components(Siewiorek and Swarz, 1982).

During the survey process the FTA user is prompted to input one or more system requirements. These requirements may be of different kinds, e. g., some may require reliability while others may require availability. FTA assumes that all stated requirements must be satisfied simultaneously by the system.

FTA's requirements survey can be reached from DesignExpert's Security Assistant (SA) (see separate documentation of SA for more information), thus coupling the two assistants together. There are two reasons to relate FTA to SA in this way. First, some users will need to address security threats that involve failures of system components. Note that FTA helps with the analysis of *uncorrelated* component failures but currently offers only limited help if failures are *correlated*, for example, as a result of a malicious attack on the system. Second, a combined FTA-SA requirements survey allows DesignExpert to present a uniform user interface. That interface elicits the system requirements by identifying the *threats* that the

system must counter, whether those threats are from malicious security intruders or random failure of computing hardware.

When used with SA, FTA offers its user explainations of the meaning of the various requirements.

3.3.2 System Model

To analyze the system requirements, the FTA user must supply an abstract model of the system. This model is hierarchical: every component and subcomponent may be composed of other subcomponents. The system model may include both hardware and software, and both hardware and software may be hierarchically decomposed. Most components will be given various attributes needed during requirements analysis.

The system model can include any of the following components and component attributes:

- hardware processors;
- hardware communication channels between processors;
- software processes;
- software communication channels between processes;
- replication factors for each process, processor, and communication channel;
- failure and repair rates for each process, processor, and communication channel;
- number of failures tolerated in each replicated ensemble;
- failure model (e.g., crash failures, Byzantine failures) assumed and guaranteed by each component;
- assignment of software components to hardware components.

In general, both hardware and software failures may be analyzed. Tolerating hardware failures is the more traditional concern, but software fault tolerance(Avizienis and Kelly, 1984) may also be analyzed in this framework[1].

The user may input the abstract model in any of several ways: textually, pictorially, or by using HTML forms[2]. Regardless of which way is chosen, a textual representation of the model is built in the Acme architecture description language(Garlan et al., 1997). We chose the Acme language to be the standard architecture description for both FTA and SA because it is a simple language designed to allow easy translation to the architecture description languages used by other tools. The Acme language also can be easily extended

[1]Currently FTA assumes that atomic components, i.e., ones with no subcomponents, fail independently. This assumption may be inappropriate, however, for software failures. Correlations between failures could be included in a future version of FTA.

[2]DesignExpert does not yet fully support architecture description using HTML forms.

by associating attributes with each component of an architecture. All of the attributes of the system model described previously are represented as Acme attributes in FTA.

The easiest way to input the abstract model in FTA is pictorially. A separate DesignExpert component, the Graphical Architecture Design Editor (GADE), is used to produce a graphical rendering of the system architecture. GADE can edit both the hierarchical structure of components within a system and the FTA attributes of these components. GADE reads and writes architecture descriptions in the Acme language.

3.4 Outputs from FTA

FTA produces three outputs:

1. a Monte Carlo simulation of the system model;
2. estimates of reliability, availability, MTTF, etc., for a given system model;
3. advice on how to correct or improve the system model.

3.4.1 Simulation

FTA's basic mechanism for requirements analysis is simulation. The Monte Carlo simulation begins with the system model. Each atomic component of the model, i.e., a component with no subcomponents and that is not assigned to another, is made to fail independently and randomly (hence the "Monte Carlo" adjective). The failure distribution is assumed to be exponential decay, taking the failure rate of each component as constant. The effect of each failure is propagated to other components: a component fails if too many subcomponent failures have happened or if it is software running on a failed hardware component. If failures propagate to the top level of the system model then the system has failed. Components may also be repaired at a constant rate. The repair of a component is propagated to other components (under the simplistic assumption that a mechanism exists to recover the state of software running on repaired hardware).

A novel aspect of FTA is that the simulation can be visualized. The GADE tool, described previously as the means for inputting the system model pictorially, can also display the steps of the Monte Carlo simulation. In this animation of the system model, GADE shows the effect of each failure and repair as it propagates through the system.

The Monte Carlo simulation is also used to estimate the reliability properties of the system. To estimate reliability, mission time, and MTTF, the simulation is run many times and the results averaged. To estimate availability, MTBF, and MTTR, a very long simulation is run in which the system can fail and be repaired many times and averages are computed from that single run. The averages computed for each of these reliability properties are uncertain because the Monte Carlo gives only a finite sample of the possible system behaviors, so FTA also estimates the uncertainty in each of these properties.

3.4.2 Design Advice

The basic FTA goal is to understand whether the system model satisfies the dependability requirements. FTA compares the reliability and availability properties and their uncertainty estimated from the Monte Carlo simulation with the requirements gotten from the user. The user is advised whether the requirements are satisfied.

Regardless of whether the requirements are satisfied, FTA may provide its user with advice on how to improve the system model. Several kinds of advice are offered:

- A critique of the attributes in the system model. FTA looks for several kinds of inconsistencies in these attributes, including: whether the fault tolerance specifications can be achieved given the model of failures (e.g., in general at least $3T + 1$ component replicas are needed to tolerate T Byzantine failures); whether obvious failure correlations exist (e.g., two replicas of the same process should not assigned to the same processor); whether the failure model makes sense (e.g., a process assumed to exhibit only crash failures should not be assigned to a processor that can exhibit Byzantine failures).

- Recommendations for increasing reliability. Improvements in reliability can sometimes be had by increasing the degree of replication and sometimes by changing the allocation of software components to processors(Nieuwenhuis, 1990). FTA estimates the potential for such improvements using a "quick and dirty" sensitivity analysis rather than using the computationally-intensive (but more accurate) Monte Carlo simulation.

- Advice about whether to run the Monte Carlo simulation longer to get better statistics. This advice is based on the estimate of the uncertainty in the reliability estimates. If the uncertainty is so large that it is unclear whether the system model satisfies the requirements then accumulating better statistics makes sense.

FTA's design advice should lead to better system models and ultimately to a sound judgement of whether particular dependability requirements can be satisfied. If a model satisfies the requirements, the number and kind of components in the model can be used to estimate the final cost of building the real system.

Chapter 4

The Security Assistant

4.1 Overview of SA

A security requirement is a written formulation of the objective of opposing some attack on, threat to or vulnerability of the given system. The Security Assistant (SA) supports the formulation of security requirements and choice of security mechanisms to meet those requirements. System security requirements derive from several sources, among which the most noteworthy are:

- Mission needs and concepts of operation
- Risk and threat assessments of potential deployment sites
- Requirements stipulated for similar systems
- Organizational security policies
- Mandatory requirements (for government systems).

SA uses a series of automated surveys to gather such information about a system under study and then analyzes that data for consistency and completeness. In addition, a deeper security analysis provides a critique of various design choices and recommends countermeasures of appropriate strength to offset known or potential risks. In this process, SA balances the cost of countermeasures and the likelihood that a threat will be actualized on the one hand, with the value of assets and the importance of the system mission on the other. The ultimate goal for both SA and the user is to characterize, or at least bound, a system's **residual risk** in a cost conscious fashion.

4.2 Classes of Users Supported

The Security Assistant provides support to users during the pre-implementation phases for new systems and maintenance and upgrade phases for legacy systems. SA support during

the requirements analysis phase deals largely with the process by which requirements are determined and formulated to counter security threats. During this phase, various groups of users benefit from security advice.

4.2.1 System Designers

System Designers benefit by including security as a system wide property entwined with architectural design and *not as an add on.* Moreover, this group benefits because Security Assistant can handle generic security concerns thus giving them the freedom to concentrate on the really unique security problems that are specific to each system.

4.2.2 System Certifiers

System Certifiers who are responsible for determining the *residual risk* associated with deploying an Information Technology (IT) system for a given mission in a given environment often have to treat each case separately. Moreover, they must contend with a mountain of required paper work as well as perform technical analyses. This group benefits by having a framework compatible with the DoD Information Technology Security Certification and Accreditation Process (DoD Instruction 5200.40,) (DITSCAP) in which to evaluate a system. A future version of SA is planned that will provide comprehensive support to DITSCAP's System Security Authorization Agreement (SSAA) as a living document and will provide certifiers with the means to assign Information Technology Security (ITSEC) classes to systems and make comparisons.

4.2.3 System Administrators

System Administrators and IT Managers benefit by having a tool that can help organize and analyze systems and networks that they are responsible for maintaining. These professional are frequently tasked with enhancing such systems by deploying new applications or hosts. This can cause security compromises for either new or old components. These decision makers, who have to bear the responsibility for the choices they make, need objective and cost conscious advice that reflects the IT system mission as well as the risks to which it is subjected. Currently, decision makers rely solely on the advice of domain experts.

4.3 System Decomposition

Modern distributed systems applications typically consist of hardware and software components that are themselves systems. This leads to the notion of "system of systems". From a security standpoint this is a very important perspective since not only must component systems be trustworthy but also they must be combined in such a way that the resulting system of systems is also trustworthy. The Security Assistant has been designed with this perspective built in.

In order to work at the system of systems level, we take the perspective that the target system under study is a set of interrelated component systems which are deployed (and possibly replicated) at sites. Viewed from the topmost level the target system has a global operational environment which is made up of the site environments. Threats to security come from these environments and can arise from natural or human sources. It is this threat space that characterizes the risks run by the target system. In addition, the target system has missions, clients, dependents and assets which play an important role in deciding the sort of security needed. Ultimately, security of the target system is achieved by determining and implementing a Security Policy which counters the threats and enables the missions while protecting assets and serving clients. Although a target system security policy is the ultimate goal, elements of it are often known in advance or are dictated by organization policy. For these reasons, several security policy elements appear as both inputs and outputs of the Security Assistant.

An important first step on the road to defining a security policy is to determine what the system wide security requirements are. This can be done by first identifying the needs that motivate the target system. Moreover, the importance of missions and the value of assets must be weighed against the cost of countermeasures to the threats in the environment.

4.3.1 Architectural Levels for Systems of Systems

In order to fully specify a target system, the Security Assistant expects the user to fill out a set of surveys for each of the four architectural levels, namely the system-wide level (aka top level), the site level, the host level, and the application level.

As indicated above, the view taken by the Security Assistant is that the top-level target system is composed of sets of components deployed at a set of sites. Roughly speaking, a site is a collection of hosts that: are usually concentrated geographically in one region, are logically related by common purpose, are physically connected to each other by a LAN, pre-date the target system, and perform functions unrelated to the target system.

Each site has an environment and a set of hosts that are connected to the site and provide the platforms onto which target system applications are deployed. Hosts are described by specifying: hardware configuration, operating system, the set of installed target system software components, and the suite of installed applications and libraries.

The application layer provides the user with a place to characterize how system resources are shared between the target system software and peer software. In addition, by providing details about the applications running of each host, many security vulnerabilities can be detected and countered during the requirements phase.

4.4 Internal Components

Security Assistant is based on data gathering technologies including a web based user interface (employing Java, JavaScript, HTML in Netscape 3.0) which is used to collect security related system design information. A multi-layered series of surveys is presented to the user

of Security Assistant. These surveys are based on HTML forms and are enhanced by means of Java and JavaScript.

Security Assistant is also built on knowledge based engineering and natural language technologies. An expert system shell called the C Language Integrated Production System(Giarratano and Riley, 1994b) (CLIPS) is used to provide analytic operations such as consistency and completeness checking of design data. In addition, security domain expertise encoded in rules is used to generate advice about the security properties of the IT system under study. In the SA system CLIPS rules representing security domain knowledge have been encoded by Key Software, Inc. There is no provision in the current prototype for users to modify this knowledge base.

Finally, Security Assistant uses a natural language text generation system (developed by CoGenTex) to create reports and summaries based on the output of the domain rules that fire in the expert shell's inference engine. In addition, summaries of the surveys themselves are produced to allow users to more easily detect oversights and errors.

4.5 Automatic Reasoning

As mentioned, SA uses an inference engine to analyze and critique user descriptions of a system under study. Knowledge of security derived from experience, publications and DoD standards such as the Trusted Computer System Evaluation Criteria(US , 1985) (TCSEC) and the Common Criteria(National Institute of Standards and Technology et al., 1996) is encoded in rules that are activated by the presence or absence of relevant facts. In addition, SA employs a *concept relation* model, Figure 4.1, on which its reasoning is based.

The arrows in the figure are labeled in such a way as to define the relationship between the concepts in the ovals. For example, **defenses** *oppose* **attack methods** and **attack methods** *cause* **damage (types)**. However, **defenses** *can be* **assets** which can *sustain* **damage (types)**.

By determining what types of damage an asset or mission can sustain, what kind of attack methods are possible, how likely it is that an attack can be mounted and what system defenses are available to counter attack methods it is possible to formulate inference engine rules that reflect rules of thumb like the following:

1. **Make sure that every asset is protected from every known potential attack method.**
2. **Balance the cost of the system defenses with the asset values and the mission importance.**

4.6 Example Use of Security Assistant

In this section we describe a scenario of using Security Assistant for a Mission Planning and Tracking System (MPATS). MPATS is an experimental secure distributed application

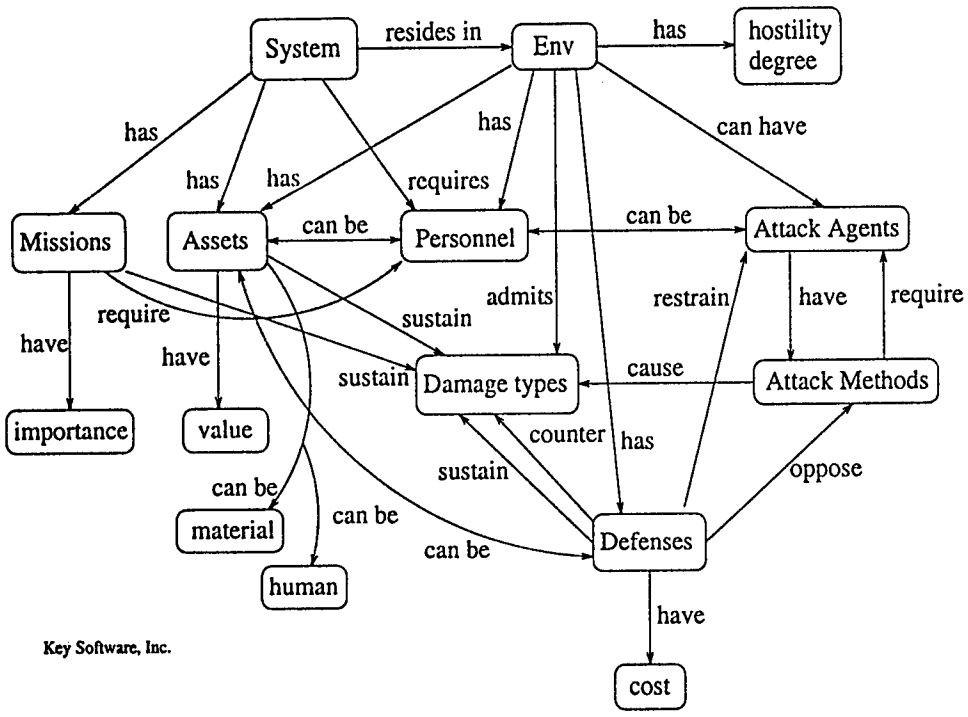


Figure 4.1: SA Concept Relation Model

(implemented but not deployed). MPATS is a typical system of systems. It consists of several components distributed across several sites (see Figure 4.2). The purpose of MPATS is to plan and track military airlift missions such as the one shown below.

The MPATS functional components are:

- FPLAN, for airlift mission planning, which deals with information on CONFIDENTIAL and SECRET levels;
- FTRACK, for airlift mission tracking which also deals with information on CONFIDENTIAL and SECRET levels;
- Wx, for weather forecasting, which deals with UNCLASSIFIED information.

These components are deployed at the following sites:

- FPLAN is deployed only at Ramstein AFB, Germany, a site which supports operation on multiple security levels, UNCLASSIFIED to SECRET;
- FTRACK is deployed on all three sites, where the sites Montijo NATO AB, Portugal and Incirlik AFB, Turkey support operation only on two levels UNCLASSIFIED and CONFIDENTIAL;
- Wx is deployed on all three sites.

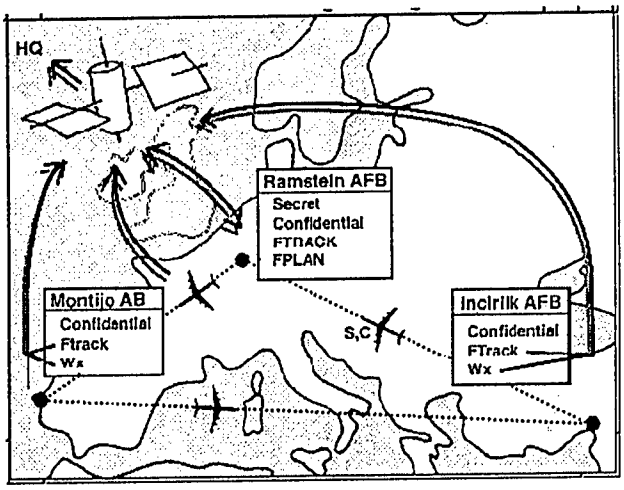


Figure 4.2: Sample MPATS System architecture

In our scenario, during the requirements analysis phase, the SA user first describes MPATS on the system-wide level, using the suite of system-wide surveys. The security environment survey is shown in Figure 4.3.

The user can validate the entered system-wide information using an automatically generated summary of this information (see Figure 4.4 for the summary of the information entered in the survey of Figure 4.3). These summaries can be called from the corresponding surveys, or the user can use a hierarchical representation of all information entered so far in order to directly access menus for generating textual summaries (see Figure 4.5). This same hierarchical representation is also useful for naviagtion purposes when a lot of information has been entered. These facilities have been designed and implemened by CoGenTex.

The system-wide survey describes the target system from the functional point of view. The deployment information is gathered by the lower level surveys: site, host, and software component.

As it was explained above, the current automated security analysis works on the system-wide level and takes into account only a part of the site security-related deployment information. The attributes that are most implortant for the automated analysis are: site network connections, security mechanisms at the site and site hostility degree.

In our scenario, from the system-wide and site surveys, SA learns that

- the Ramstein site has data at multiple levels,
- that for the flights database asset of the Fplan functional component deployed at Ramstein disclosure, descruction/corruption, and substitution damage types are of concern;
- that Ramstein and one other site, Montijo, are in benign environments, while the third site, Incirlik, is in a potentially hostile environment;
- that Ramstein and Incirlik sites can communicate over the network;

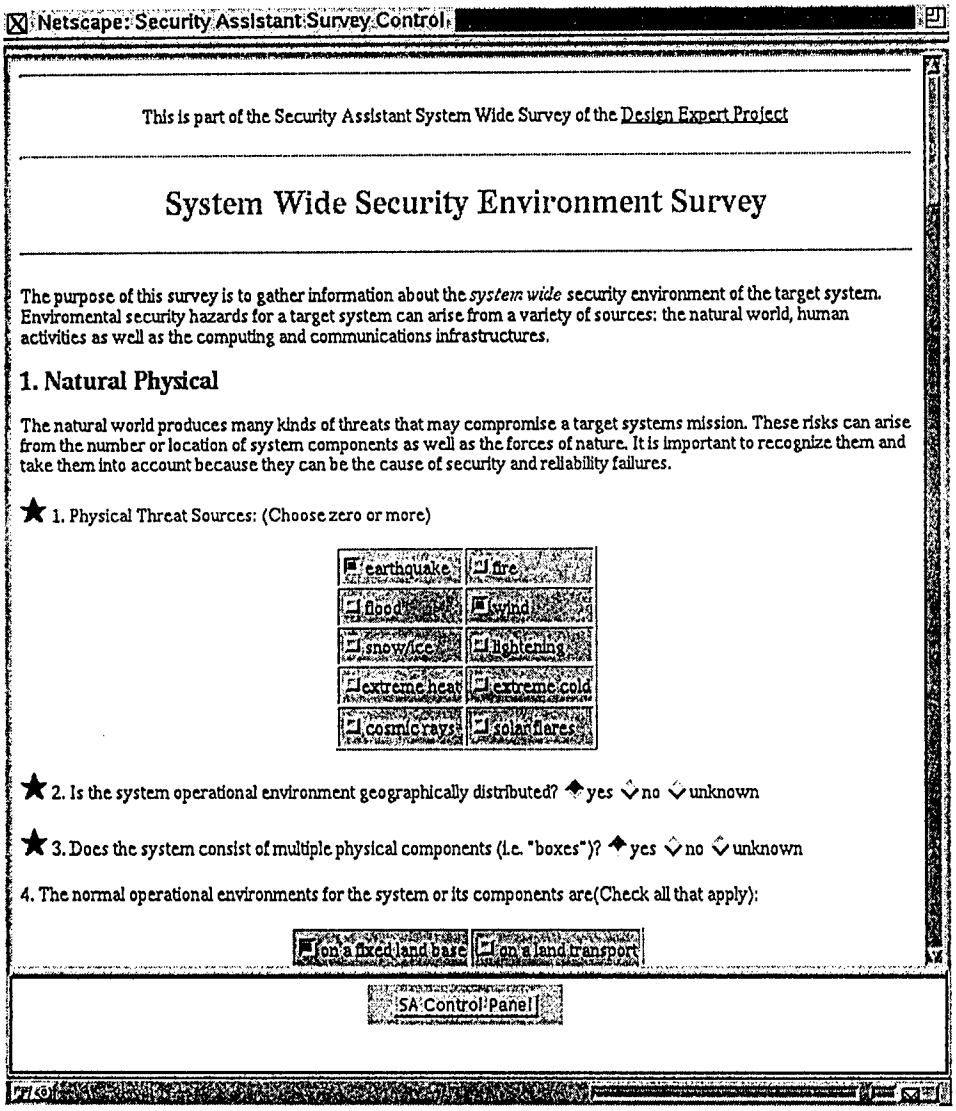


Figure 4.3: SA survey screen for system-wide security environment (excerpt)

- that users at any site have a global identity allowing them access to the other sites on the network; and
- that users at Incirlik use passwords as an authentication mechanism.

Based on this information, SA recommends installing non-discretionary security measures at Ramstein and requiring stronger individual authentication at Incirlik and Montijo (see Figure 4.6). he vulnerability in question is that, because of its greater hostility level, hostile elements at Incirlik or Montijo couls potentially use stolen passwords and attack the assets at Ramstein, using the user global identity feature of the MPATS system. The automatically generated explanation of the reasoning underlying this recommendation is shown in Figure 4.7. The DesignExpert explanation facility was designed and implemented by CoGenTex; for a fuller presentation, see Chapter 5.

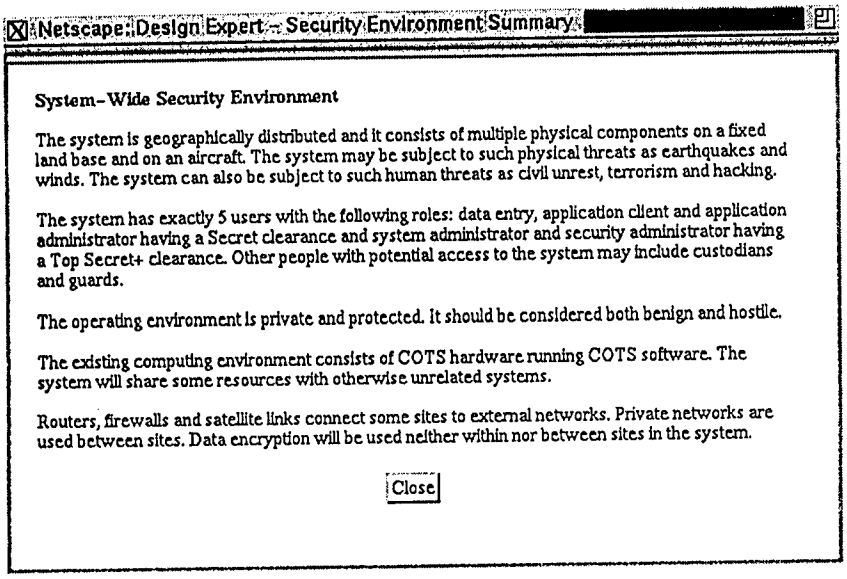


Figure 4.4: Summary of SA survey for system-wide security environment

If more detailed deployment information is available, the user can use host and software component surveys to enter this information into SA's repository. Figure 4.8 shows an excerpt from the Host survey, and Figure 4.9 the correpsonding summary.

4.7 Summary and Future

The research prototype of Security Assistant demonstrates while many features could be more strongly represented in a robust finished product, our basic goal has been largely realized. Our objective was to develop a security tool on top of an underlying methodology that would, in principle, be able to:

- Provide a comprehensive framework in which the system wide security properties can be described;
- Provide a means to analyze and critique security relevant design choices;
- Identify requirements and give advice on how design choices may be improved;
- Suggest mechanisms to counter threats based on encoded knowledge in addition to input provided by the user;
- Balance cost of security measures with system mission importance in the context of a realistic threat assessment;
- Handle legacy systems as well as new designs;
- Handle systems of systems and the environments into which they are deployed;

- Explain reasoning that is used to analyze and give advice;
- Provide a means to automatically generate reports and summary documents.

Security Assistant would benefit from certain new features. For example, a connection to or synopsis of the CERT and FIRST databases of flawed software would be a big improvement. Focusing on DITSCAP would make the tool more relevant to real users in DoD and its contractor community.

The use of rule based reasoning in the security domain can be applied not only to systems like SA but also to network monitoring. Software agents working cooperatively throughout a network can be tasked to gather, pool and analyze security properties using Java based versions of CLIPS such as Jess.

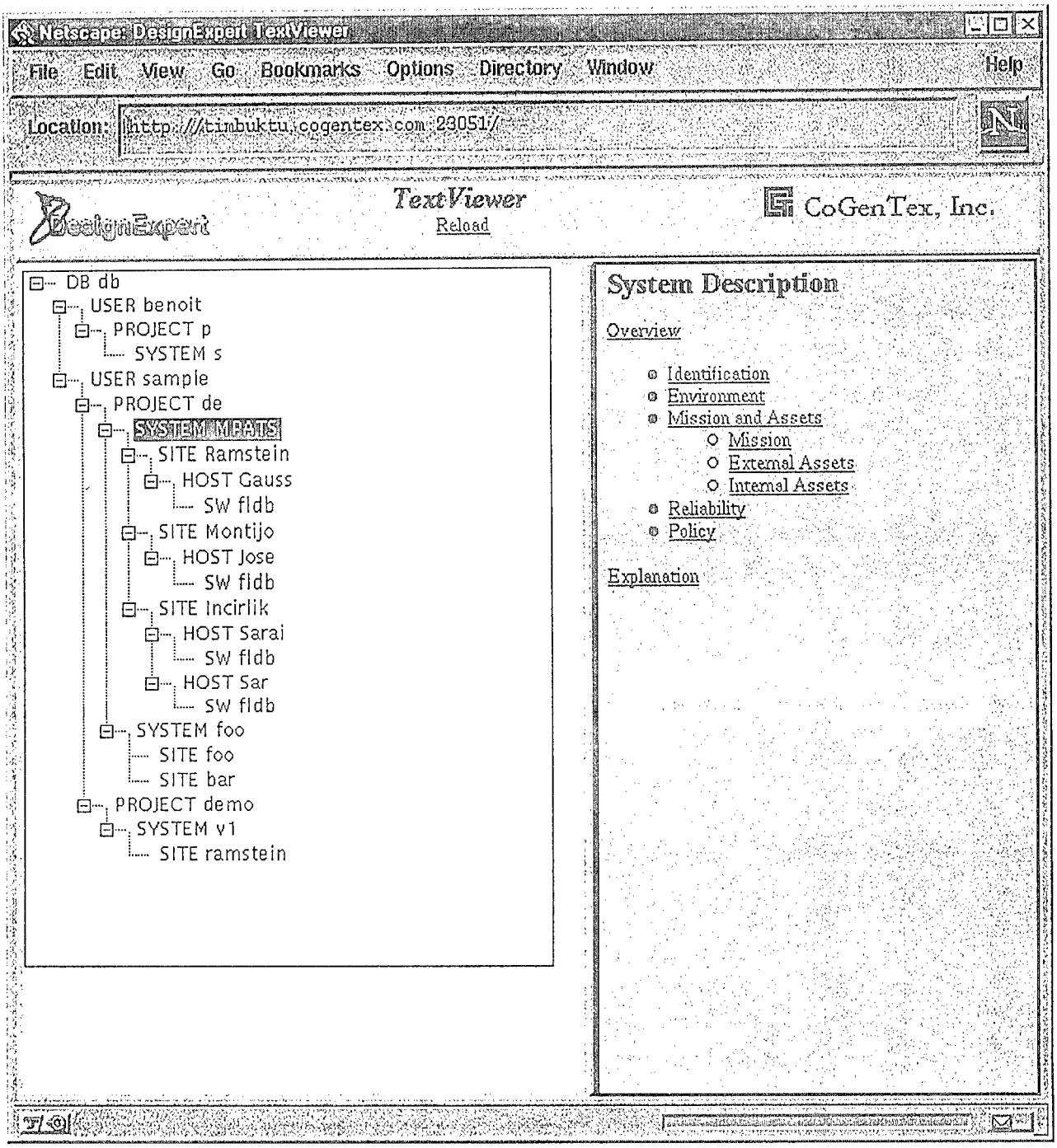


Figure 4.5: Survey Navigation Tool (left panel) with access to textual summaries (right panel)

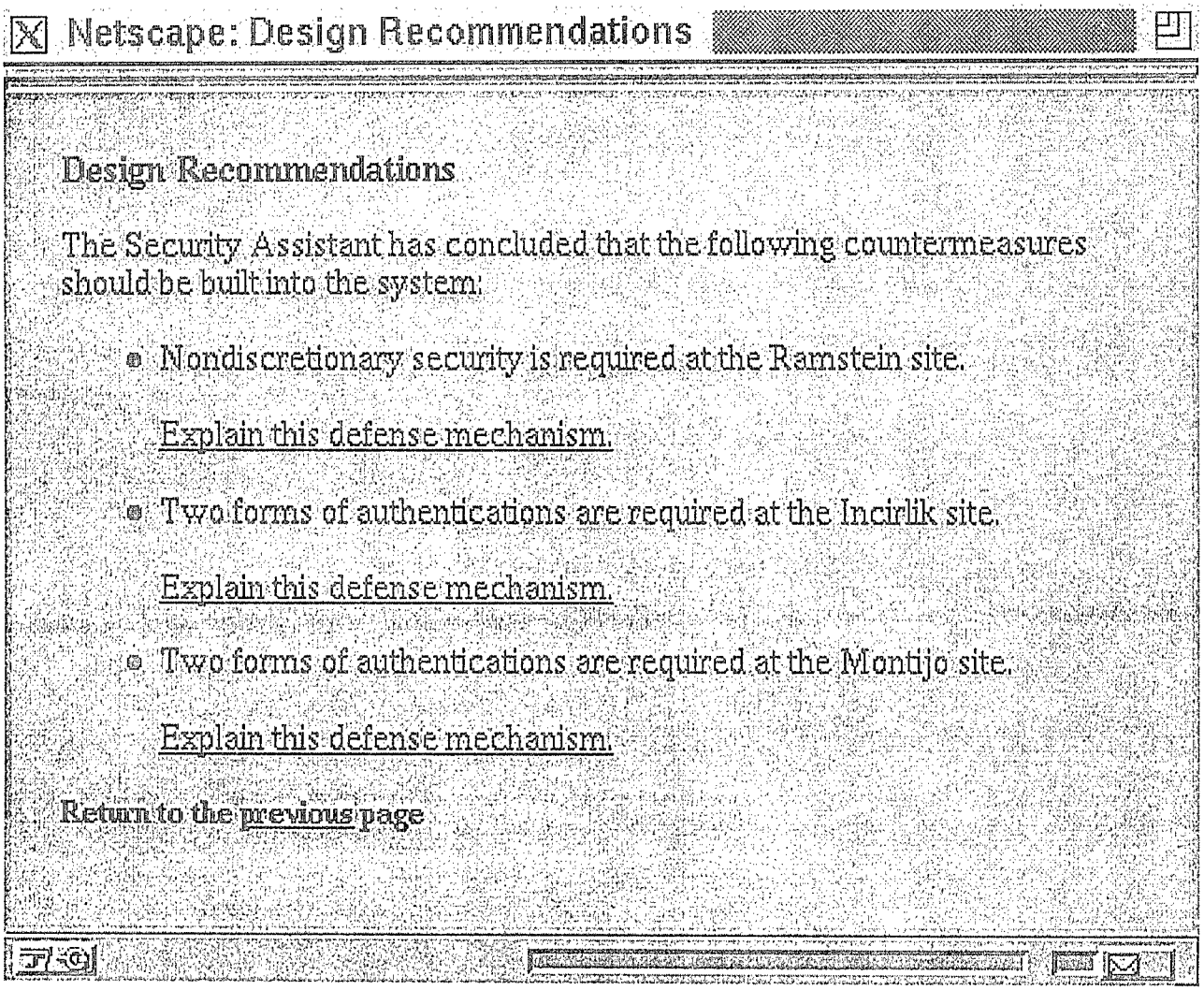


Figure 4.6: SA recommendations for MPATS system

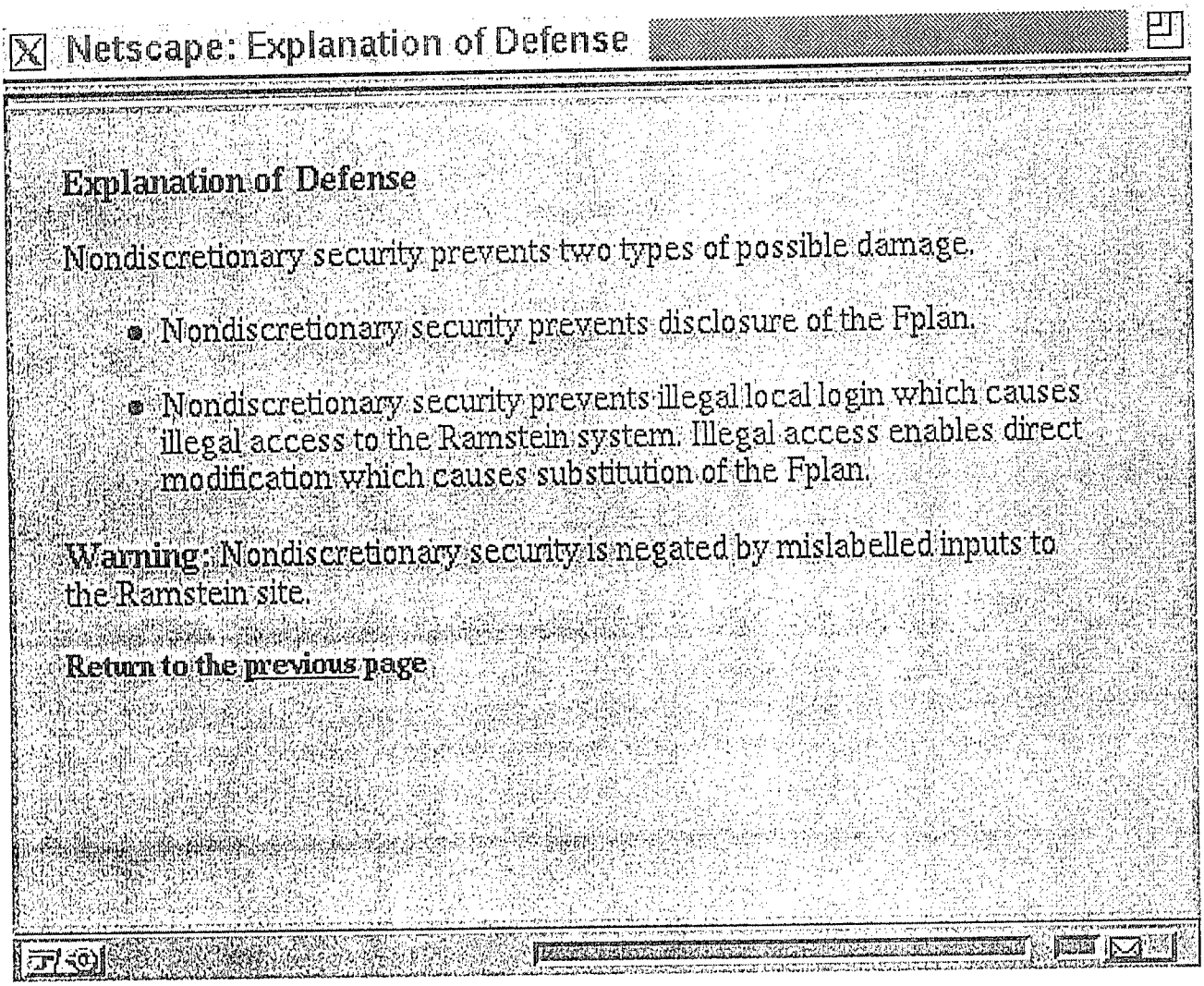


Figure 4.7: Automatically generated explanation of SA recommendation

4. Host Security Attributes

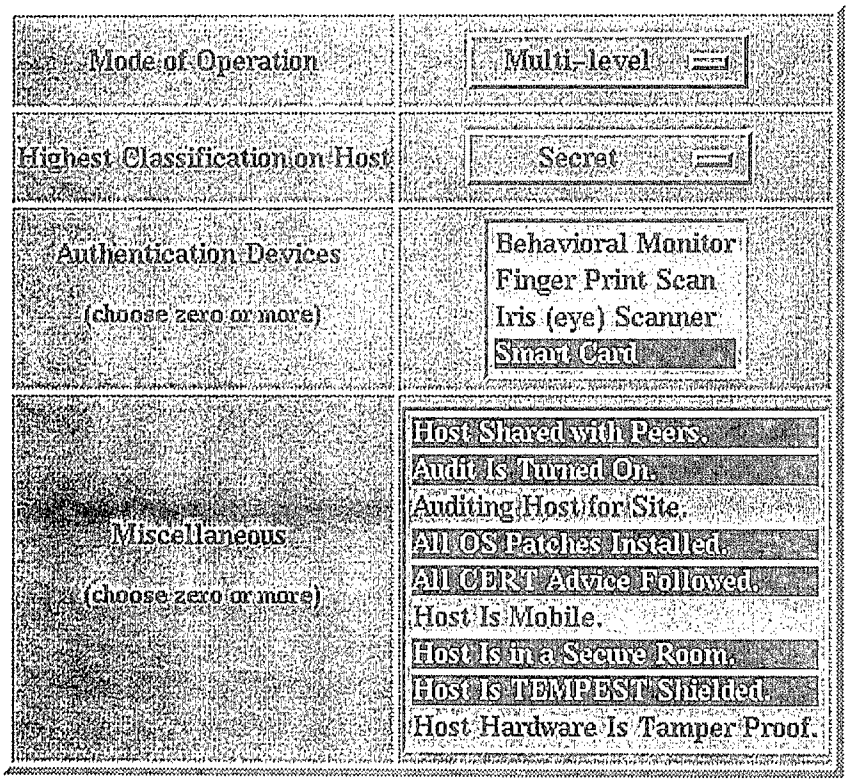


Figure 4.8: SA survey screen for the host (excerpt)

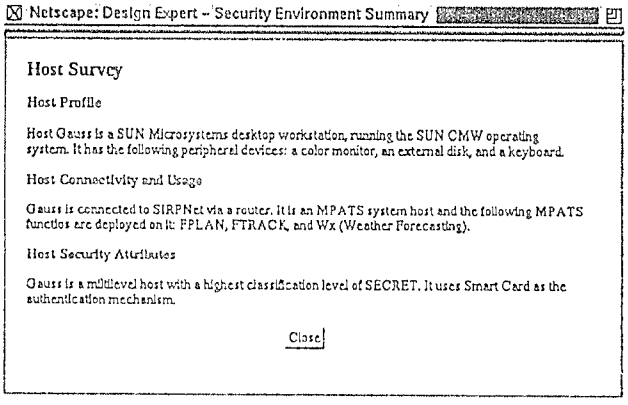


Figure 4.9: Summary of SA survey for a host

Chapter 5

Automatic Explanation Generation in DesignExpert

5.1 Expert System Explanation

Expert systems were one of the first applications to emerge from initial research in artificial intelligence, and the explanation of expert system reasoning was one of the first applications of natural language generation. This is because the need for explanations is obvious, and generation from a knowledge-based application such as reasoning should be relatively straightforward. However, while explanation has been universally acknowledged as a desirable functionality in expert systems, natural language generation has not taken a central place in contemporary expert system development. For example, a popular text book about expert systems such as (Giarratano and Riley, 1994a) stresses twice in the introduction the importance of explanation, but provides no further mention of explanation in the remaining 600 pages. (The book is based on the popular CLIPS framework.) In this paper, we present a new approach to enhancing an expert system with an explanation facility. The approach comprises both software components and a methodology for assembling the components. The methodology is minimally intrusive into existing expert system development practice.

This chapter is structured as follows. In Section 5.2, we discuss previous work and identify shortcomings. We present our analysis of knowledge types in Section 5.3. Section 5.4 discusses the relation between the Security Assistant and its explanation facility. Finally, we sketch a general methodology for explainable expert system engineering in Section 5.5.

5.2 Previous Work

A very important early result (based on experiences with explanation[1] in systems such as MYCIN (Shortliffe, 1976)) was the finding that "reasoning strategies employed by programs do not form a good basis for understandable explanations" (Moore, 1994, p.31). Specifically, simply paraphrasing the chain of reasoning of the expert system does not let a human user easily understand that reasoning.

Two separate approaches have been proposed to address this problem:

- In the **Explainable Expert System** (EES) approach (Swartout et al., 1991; Swartout and Moore, 1993), the knowledge representation used by the expert system is enriched to include explicit "strategic" knowledge, i.e., knowledge about how to reason, and domain-specific knowledge. From this knowledge, the rules used by the expert system are compiled, and this knowledge is also used to provide more abstract explanations of the system's reasoning.
- In the **Reconstructive Explainer** (Rex) approach (Wick, 1993), the expert system is unchanged, but after it has performed its reasoning, a causal chain for explanation is constructed from the input data to the conclusion reached previously by the expert system as a separate process. The work of (Tanner et al., 1993) can also be seen as falling in this paradigm, since a separate representation of knowledge (the "functional representation") is used only for explanation, and the explanation must be specially derived from this.

These approaches have in common a preoccupation with a categorization of knowledge used in the system into different types. EES concentrates on an abstract representation of strategic knowledge (how does a particular action of the system relate to the overall goal?) and on the representation of design rationale (why are actions reasonable in view of domain goals?). In addition, there is terminological domain knowledge (definitions of terms). Rex and related approaches have a representation of domain knowledge, along with domain rule knowledge (mainly causality), which is completely separate from that used by the expert system itself. This knowledge is used to derive an "explanation path" through the domain knowledge representation.

There are problems with both approaches. EES has not proven to be a fully satisfactory solution to the problem of expert system explanation. The problem is that the writers of expert systems have not been too quick or too eager to adopt frameworks such as EES. The requirement for a more abstract representation of knowledge (from which the actual expert system rules are compiled) that EES imposes may be considered onerous by the expert system developer, appearing unmotivated from the point of view of the core functionality of

[1] We do not consider explanation generation from data bases (for example, (McKeown, 1985; Paris, 1988; Lester and Porter, 1997)) to be the same problem as expert system reasoning explanation (even though we may use some similar techniques). In data base explanations, the knowledge to be communicated is static and its representation is given *a priori* as part of the statement of the generation problem. In expert system explanations, the knowledge to be explained is generated dynamically, and the proper representation for this knowledge is part of the solution to the problem of expert system explanation, not its statement.

the system, namely reasoning (as opposed to explanation). Presumably, it is difficult for one and the same person to be a domain expert and a expert on communication in the domain.

In the Rex approach, the obvious problem is that in order to generate an explanation, additional reasoning must be performed which in some sense is very similar to that done by the expert system itself (e.g., finding causal chains). This is redundant, and does not result in a clear separation between reasoning and explanation. While Wick (1993) argues against such a separation on philosophical grounds, practical constraints suggest, as indicated above, that the domain expert responsible for implementing the reasoning system should not also be responsible for implementing the explanation capability, and that the communication engineer (responsible for implementing the explanation facility) should not need to replicate domain reasoning.

In this paper, we present a new approach (architecture and methodology) to expert system explanation which does not require the expert system writer to take into account the needs of the explanation while writing the rules. At the same time, we avoid the necessity of having a separate domain reasoning component for the explanation generation. Instead, the expert system is largely considered a stand-alone application, onto which explanation is added. However, this is done by having a communication engineer design a second knowledge representation (separate from the expert system's domain knowledge representation) specifically for the purpose of communicating explanations. This representation is instantiated by the expert system as it reasons, not by a separate module after reasoning has occurred. Thus, no separate reasoning facility is needed.

5.3 Types of Knowledge in Explanation

We follow previous work in distinguishing different types of knowledge. However, we use operational criteria: we classify knowledge by what it is used for and who is responsible for its engineering, not by its structure or contents. We briefly present our classification here and illustrate it on an example in the following section.

- **Reasoning domain knowledge** (RDK). This is knowledge about the domain needed to perform the reasoning. Typically, it includes rules, terminological knowledge, and instance knowledge. It is encoded by the domain expert in the expert system proper.
- **Communication domain knowledge** (CDK). This is knowledge about the domain which is needed for communication about the domain. It typically is a different "view" on the domain knowledge than RDK, and may include additional information not needed for the reasoning itself. It is encoded by the communication engineer in the explanation facility.
- **Domain communication knowledge** (DCK). This is knowledge about how to communicate in the domain. DCK typically includes strategies for explanation in the given domain, and knowledge how to describe the entities of the domain. It is encoded by the communication engineer in the explanation facility.

The distinctions may at first seem overly fine-grained. However, each type of knowledge is distinguished from the other types. CDK is domain knowledge, but it is domain knowledge that is needed only for communication, not for reasoning (as is RDK). RDK and CDK of course overlap, but they are not identical. This is in fact the lesson from much previous work in expert system explanation, for example the work of Paris et al. (1988) contrasting "the line of reasoning" and "the line of explanation", and the claim of Swartout et al. (1991) that the domain representation must be augmented with additional knowledge about the domain and about reasoning in the domain. Many researchers have identified the need for packaging domain knowledge differently for communication. For example, the "views" of Lester and Porter (1997) can be seen as a form of CDK, though they are not a declarative representation. What is new in our work, however, is the proposal that CDK should be represented explicitly in a distinct representation from the domain knowledge.[2]

CDK is different from DCK in that CDK is knowledge about the domain as it is needed for communication, but DCK is knowledge about how to communicate in that domain (and in a specific communicative setting characterized by factors as diverse as communication type or genre, hearer needs, communication medium, or cultural context). Therefore, for expert system explanation applications, CDK is conceptual knowledge (what conceptual content must be conveyed to the user to explain system reasoning effectively?), while DCK is knowledge about language use (how do we use linguistic acts to explain system reasoning effectively?).[3] DCK may be expressed in communicative plan operators which achieve goals related to the hearer's cognitive state, while CDK would never include plan operators related to the hearer's cognitive state because the hearer is not part of the domain of the expert system.

5.4 The Security Assistant

The Security Assistant or SA (see Chapter 4 and also (Webber et al., 1998)) is part of the DesignExpert tool which helps software engineers analyze system-wide (or "non-functional") requirements such as security, fault-tolerance, and human-computer interaction. The SA

[2]While CDK is closely related to *content selection*, it should not be equated with content selection, which is often seen as the first task in text planning (followed by content ordering). Content selection is entirely oriented towards the anticipated act of communication, and hence defined by its parameters: what the communicative goal is, what the medium is, who the hearer is, and other constraints (length of communication, and so on). CDK is knowledge needed for content selection, but excludes all choices that depend on knowledge of the intended act of communication. For example, CDK might include relative salience between domain objects, but does not include information about how salient an object needs to be in order to interest the hearer. However, we admit that the distinction may be blurred, especially in implementations.

[3]While DCK is domain- and genre-specific knowledge about how to communicate, we do not claim that the same type of reasoner with different domains (say, an expert system for car repair and an expert system for helicopter repair) would necessarily require different DCK. However, the type of expert system in the two cases might be very similar, and it is this fact that would allow us to re-use the same DCK. Thus, from the point of view of the explanation system, the "domain" is not the domain of the expert system, but the type of the expert system. For a discussion of the distinction between domain communication knowledge and domain-independent communication knowledge, and for an argument in favor of the need for DCK, see (Kittredge et al., 1991).

aids a software engineer in choosing security measures to protect valuable system assets (e.g. important data) against likely threats (e.g. disclosure or corruption). In the following three subsections, we discuss how the three types of knowledge discussed in the previous section – RDK, CDK, and DCK, are represented and used in the SA.

5.4.1 The Expert System: Reasoning Domain Knowledge

The SA first queries the user for information about entities of the system to be analyzed, such as system assets, system components, and system sites, and the damage types that are of concern for these entities. Additional damage types are inferred for each important asset of a system (e.g. data can suffer from disclosure or corruption). The system then reasons about possible defenses that directly prevent these damage types. If no single complete defenses can be found, the SA determines all attack methods which can cause the damage, and then deduces all enabling conditions for such attacks. It subsequently determines defenses that prevent such enabling situations. This reasoning can then be iterated. The result of the SA's reasoning is a list undefended assets and, for each such asset, a a list of recommended defenses.

For example, suppose direct modification by a malicious user has been identified as a possible damage to a system asset (say, a database), and that the SA can determine no immediate defense against direct modification (for example, it is impossible to disable all editors). Modification is only possible after the malicious user has gained illegal access to the system. In this case, we would say that illegal access *enables* modification. A defense against illegal access is therefore also a defense against modification.

The knowledge needed for reasoning is expressed in the usual manner as production rules which, if the conditions are met, assert the existence of new damages, defenses, enabling conditions, and so on.

5.4.2 The Content Representation Graph: Communication Domain Knowledge

In SA, the starting point for expressing CDK is a domain model of the type that is used in object-oriented design and analysis. Our domain model (Figure 5.1) represents security domain concepts, various attributes and concept relationships, as they are used in explanation. The domain model was created by analyzing how a domain expert would explain the reasoning of the SA to non-experts, using a small corpus of explanations. Each of the boxes in the model stands for a concept in the security domain, and inside these boxes are attributes associated with the concept. Arrow-tipped edges represent relations between concepts in the domain model Database, triangle-tipped edges represent *is-a* relations and diamond-tipped edges are *has-a* relations. Some examples:

- *Defense* objects have *id* (name) and *cost* attributes;
- *Damage* objects have *id*, *severity* and *type* attributes;

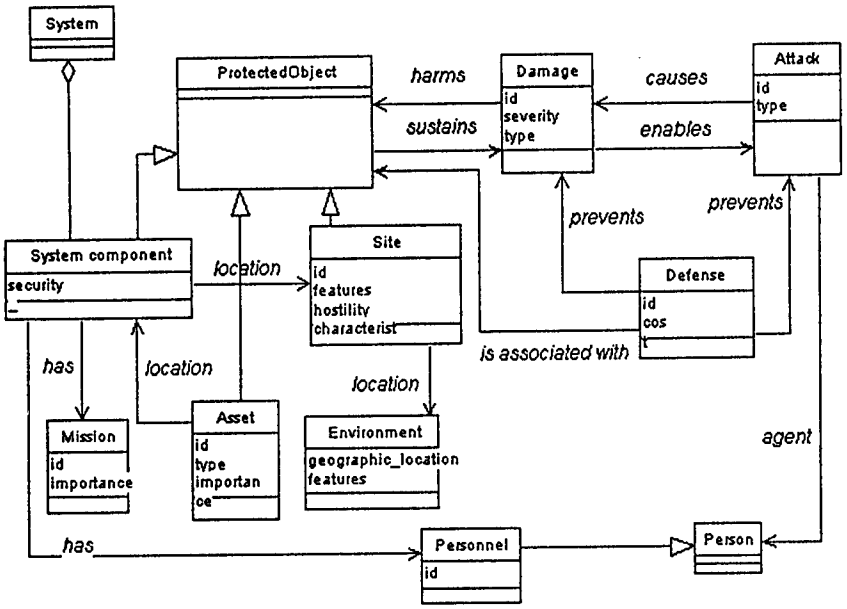


Figure 5.1: The domain model

- *prevent* is a relation that holds between a *Defense* instance and a *Damage* instance;
- *Site*, *Asset* and *System component* are different sub-classes of *ProtectedObject*;
- A *System* consists of one or more *System components*.

The CDK expressed in this domain model has no role in the expert system reasoning. In fact, during the reasoning process, the expert system models the relations as primary objects, and the concepts of our domain model are merely slots of the relations in the expert system. As a result, the relations typically are not binary, but n-ary. In contrast, the domain model contains only binary relations. This reflects, we claim, the difference between the optimal way of representing knowledge for machine reasoning, and the way in which humans model the world (which is what the CDK domain model captures). As an example of the difference in relations, the relation that corresponds to the CDK domain model's *prevent* relation between *Defense* and *Damage* corresponds to, in the reasoning component, a quintary relation between the defense, the location of the defense, the damage it prevents, the locations at which it prevents the damage, and the damages that negate the defense. Another example is the *likely_attack_method* relation used in RDK (and its structural clone, the *possible_attack_method* relation) of the reasoning component, which is a ternary relation between an asset, a location, and an attack method. As can be seen from the domain model diagram, this relation is not modeled in CDK at all.

Knowledge about domain concepts and relationships is not sufficient for generating an expressive explanation. Additional CDK is required in order to select and organize content according to explanation type and length limitation. The domain model is therefore augmented with the following information.

- Importance level, which is defined for every relation and attribute. This information about relative importance of attributes and relations enables us to produce explanations of different length. For example, the relation *prevent* between *Defense* and *Damage* has higher importance level than the relation *have* between *SystemComponent* and *Mission*. In our domain model, we use a two-valued scale.

- A key attribute for each concept which is required in instances of the concept and which identifies an instance of the concept. For example, *id* is a key attributes for *Site* but *HostilityCharacteristic* is not a key attribute.

- Mutual dependencies among concept relations and attributes. This information covers cases in which a particular relation or attribute can be presented only if some other relations or attributes are presented. For example, the relation *prevent* between *Defense* and *Attack* should be included only if the relation *cause* between *Attack* and *Damage* is included as well.

- Order among relations and order among attributes of the same concept, namely in what order should relations of the concept be presented, e.g. for concept *damage* arc *goal* is ordered before arc *enable*.

- Meta-relations between relations of the same concept. For example, there is a meta-relation purpose between (*Defense prevent damage*) and (*Defense is associated with ProtectedObject*).

To derive the CDK needed for a specific explanation task, the augmented domain model is instantiated. While the reasoning component performs the reasoning proper, it also populates the concepts of the augmented domain model with instances. The result is an instantiated model that contains concept instances, and attributes bound to their values. We called this instantiated model the "content representation graph" (CRG) of the explanation. The CRG contains all the information that is needed for the generation of the explanation text. An example of a CRG is shown in Figure 5.2.

5.4.3 Text Planning: Domain Communication Planning

As already mentioned, the CRG does not determine the form of the text, but only restricts its content. We implemented two different text types that build different text plans (and hence different texts) from the same CRG. The first type is intended to be used in an interactive setting, where the user can request more information if he or she is interested in it, by clicking on hyperlinks. An example is shown in Figure 5.3, where hyperlinks are shown by underlining.

However, for the DesignExpert application it is also necessary to generate explanations that are free of hypertext for inclusion in printed documents. These texts must include the entire explanation at a level of detail appropriate for the kind of expected reader. An example is shown in Figure 5.4. In order to create hyperlink-free explanation text, the CRG must be traversed according to constraints at every nodes: which attributes to use to describe the

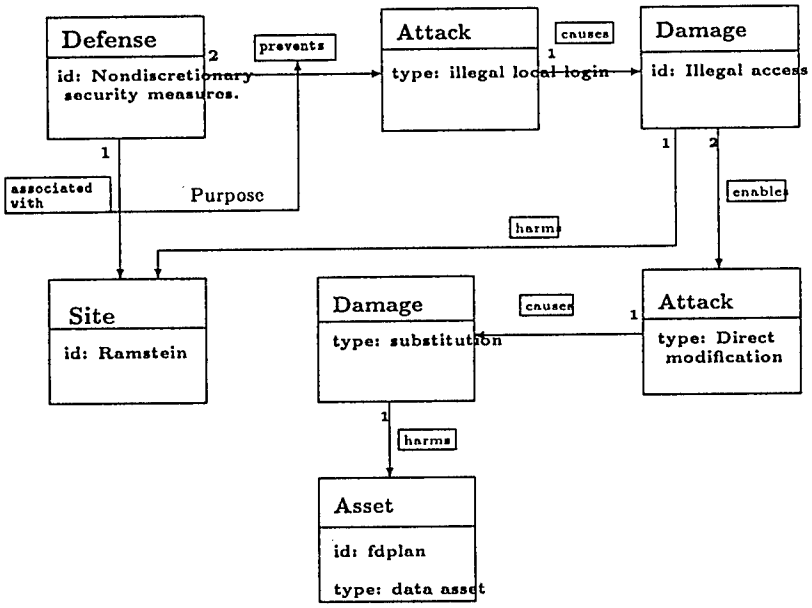


Figure 5.2: The content representation graph (instantiated domain model) for the example, representing the full CDK

Nondiscretionary security measures are required on the Ramstein site.

- Which damage do nondiscretionary security measures prevent?
- Which assets do nondiscretionary security measures protect?

Figure 5.3: The interactive hypertext

Nondiscretionary security measures are required on the Ramstein site in order to prevent substitution of data asset "ftdplan". These measures prevent substitution because they prevent illegal local login to the Ramstein system, which may enable illegal access. Illegal access may enable direct modification of data asset "ftdplan", and direct modification may cause substitution.

Figure 5.4: The fluent, hyperlink-free text

object, which relations of this object with other object must be presented in explanation, in what order to present the relations and what are meta-relationship between them. The planner processes every graph edge according to specified order, and structures resulting phrases with respect to meta-relations.

For both text types, we used a text planner with a declarative formalism for text plan specification, which directly expresses the DCK (Lavoie and Rambow, 1998). Other representations of domain-specific text planning knowledge could also have been used, and we omit details of the formalism here.

5.5 Methodology

We propose the following methodology for developing an explainable expert system. We assume three roles, that of the domain expert (where "domain" refers to the domain of the expert system, such as computer security or infectious diseases), knowledge engineer (a specialist in eliciting and representing domain models, specifically in the form of expert systems), and a communication engineer (a specialist in analyzing and representing the knowledge needed for efficient communication).

1. The knowledge engineer creates the expert system in consultation with the domain expert, using any sort of tool or shell and any sort of methodology that are convenient.

2. The domain expert writes several instances of (textual) explanations of the type needed for the application in question, based on scenarios that the expert system can handle.

3. The communication engineer analyzes the corpus of hand-written explanations along two lines:

   - The domain concepts that are reported in the text are analyzed and recorded using an object-oriented modeling technique, perhaps augmented by more expressive constructs, such as meta-relations (relations between relations). This structure is the content representation graph, represents the CDK (both the augmented domain model and its instances).
   - The structure of the text is recorded using some notation for discourse structure suitable for the text planner being used in the text generator (say, RST (Mann and Thompson, 1987)).

4. Using the CDK representation, the communication engineer consults with the domain expert and the knowledge engineer to define a mapping from the domain representation used by the expert system to the CDK domain model devised by the communication engineer. The communication domain knowledge representation may be modified as a result.

5. The knowledge engineer adds rules to the expert system that instantiate the communication domain knowledge representation with instances generated during the reasoning process.

6. The communication engineer designs a text planner that draws on the knowledge in the CDK representation and produces text. This task involves the creation of an explicit representation of DCK for the domain and task (and genre) at hand.

The resulting system is modular in terms of software modules. The expert system is preserved as a stand-alone module (though its rule base has been somewhat extended to identify communication domain knowledge), as is the text planner. Both modules can be off-the-shelf components. Only the CDK representation is designed in a task-specific manner, but of course standard knowledge representation tools, object-oriented data bases, or the like can be used.

In addition, the methodology is modular in terms of tasks and expertise. The domain expert and knowledge engineer do not need to learn about communication, and the communication engineer does not need to understand the workings of the expert system (though she does need to understand the domain well enough in order to design communication strategies for it, of course).

5.6 Conclusion

We have developed an approach to expert system explanation which is based on a classification of types of knowledge into reasoning domain knowledge, communication domain knowledge, and domain communication knowledge. We have argued that this distinction, in addition to being theoretically appealing, allows us to better manage the software engineering aspect of explainable expert system development.

While we think that our approach is well suited to explaining the reasoning of expert systems to users after the fact, the approach does not, at first glance, appear to lend itself very well to answers to "Why are you asking?" type questions from the user (as opposed to "Why are you recommending this?", which is what the SA answers). This is because the CDK is not intended to mimic the system's reasoning. However, it may be possible that the same kind of CDK can be used to answer questions before the reasoning is complete. We hope to investigate this in future work.

Chapter 6

The Human-Computer Interaction Assistant

6.1 Overview of HCIA

The goal of the HCIA research and development project (originating in its predecessor, DesignPro) has been to provide the following support to developers during system user interface development life cycle:

1. For new or evolving capabilities of the system, as part of the requirements analysis, HCIA elicits from the developer various characteristics of the capabilities in question and of goals of the system as a whole that are relevant to HCI, and then provides HCI expert advice on the user interface with the system in the light of the elicited characteristics of the system and problem domain.

2. For the prototyped or implemented user interface, as part of the system evaluation, HCIA provides an assessment and a critique of the interface, taking into account the goals of the system, the HCI-relevant characteristics of the system elicited during the requirements analysis and the given advice.

The HCIA has two components, each corresponding to one of the two tasks above. The **HCIA Advisor** addresses the first task above. It was designed and implemented by Drexel University (under the supervision of Dr. Lee Ehrhart, following previous work under the name DesignPro) and later reimplemented and improved by CoGenTex. The **HCIA Critic** addresses the second task above. It was designed and implemented by CoGenTex with support from Dr. Scott Overmyer of Drexel University. This critiquing component focuses on HTML-based interfaces. The following two sections present these two systems in turn. The final section of this Chapter contains a summary of the guidelines that we have used in developing the HCIA Critic.

6.2 The HCIA Advisor

At the core of the Advisor facility of the Human-Computer Interaction Assistant (HCIA Advisor) is the concept of "interaction" with computers, information, problems, algorithms, displays, etc. We regard interaction as any exchange between human and digital-electronic systems. Using the "interaction" paradigm, the HCIA Advisor knowledge base synthesizes findings from research in the cognitive sciences with knowledge about the information display and interaction technologies to support human-computer interaction analysis, prototyping, and evaluation.

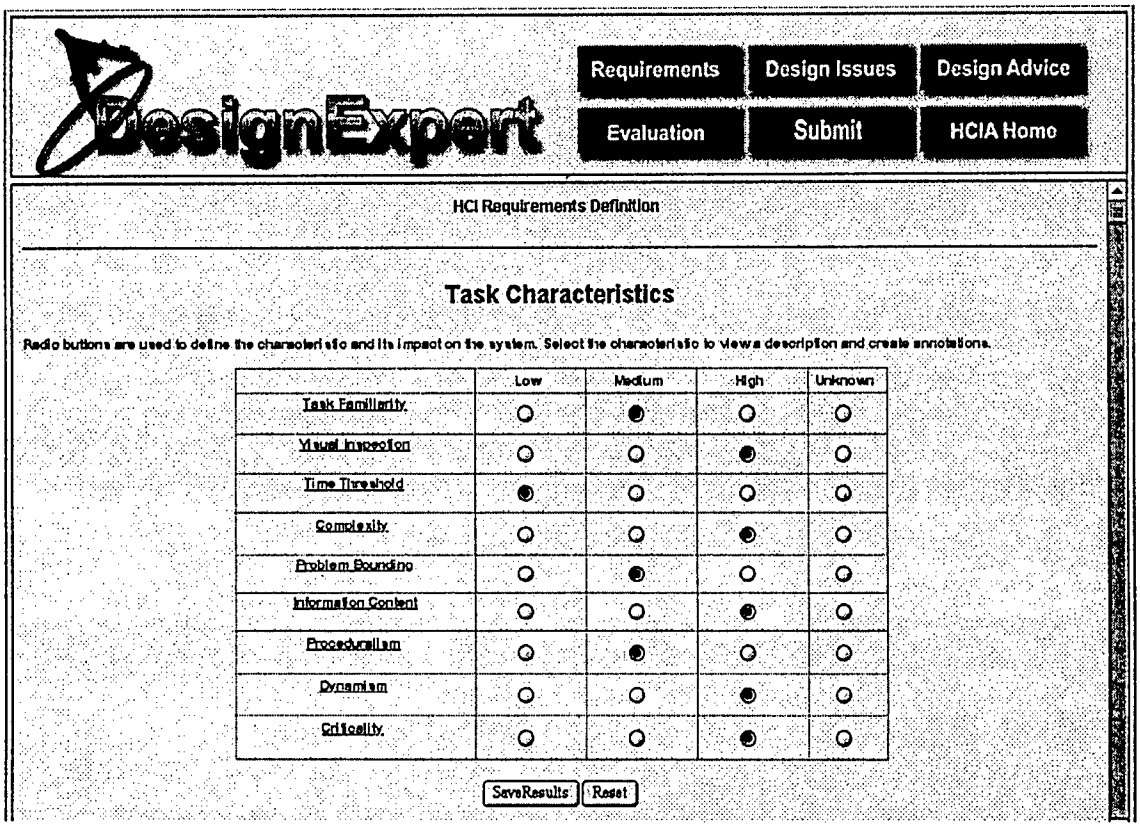


Figure 6.1: Sample HCIA Advisor questionnaire for task characteristics

The HCIA Advisor user first identifies the kinds of cognitive tasks the users may be required to perform and examine the factors affecting performance. If a task affects decision performance, it is necessary to find out what characteristics of the task do so. Meister (1981) identifies five task dimensions that may affect performance:

- Functional requirements (cognition, perception, etc.)
- Complexity
- Mental workload
- Temporal factors (pace, duration, sequence, etc.)

- Criticality

Cognitive task taxonomies, such as those found in (Fleischman and Quaintance, 1984) and (Rasmussen et al., 1990) can be used as a filter to identify and categorize basic cognitive tasks with respect to these dimensions. In addition, Andriole and Adelman (1995) present a taxonomic discussion of human information processing and inferencing tasks with respect to the potential cognitive errors associated with each. These sources and others were used in the design of the HCIA Advisor to define a set of task characteristics (e.g., task complexity, etc.) for which the user sets parameter values using a simple categorization of High-Medium-Low.

More precisely, the HCIA Advisor queries the user for information pertaining to four areas:

- users — experience, training, organizational roles;
- tasks — high-level functions, performance goals, decision task characteristics (e.g., timing, criticality, etc.);
- organizational context — organizational goals, missions, control structures, communication modes, "culture";
- and environmental context — when, where, how, and under what conditions the system will be used.

In addition, the HCIA Advisor requires the user to further bound the design options by identifying:

- any system-level constraints (such as hardware platform, storage limitations, operating system, etc.);
- and project variables (such as contract flexibility and project team composition).

To aid in this definition, the system provides descriptions of each task characteristic and an explanation of the criteria for assigning the parameter value. Space is also provided for user annotation to explain the designer's rationale for the rating given the characteristic in the context of the project. As an example, the HCIA Advisor questionnaire for task characteristics is shown in Figure 6.1.

Once the profiling is accomplished and constraints are specified, the analyst submits the requirements to the HCIA Advisor knowledge base for design suggestions regarding technologies and techniques for user-machine dialog, information presentation, and other interface and interaction design features. HCIA Advisor presents design advice for specific technologies and techniques in three categories.

1. data and information coding;
2. display design and features;
3. and dialog and interaction styles

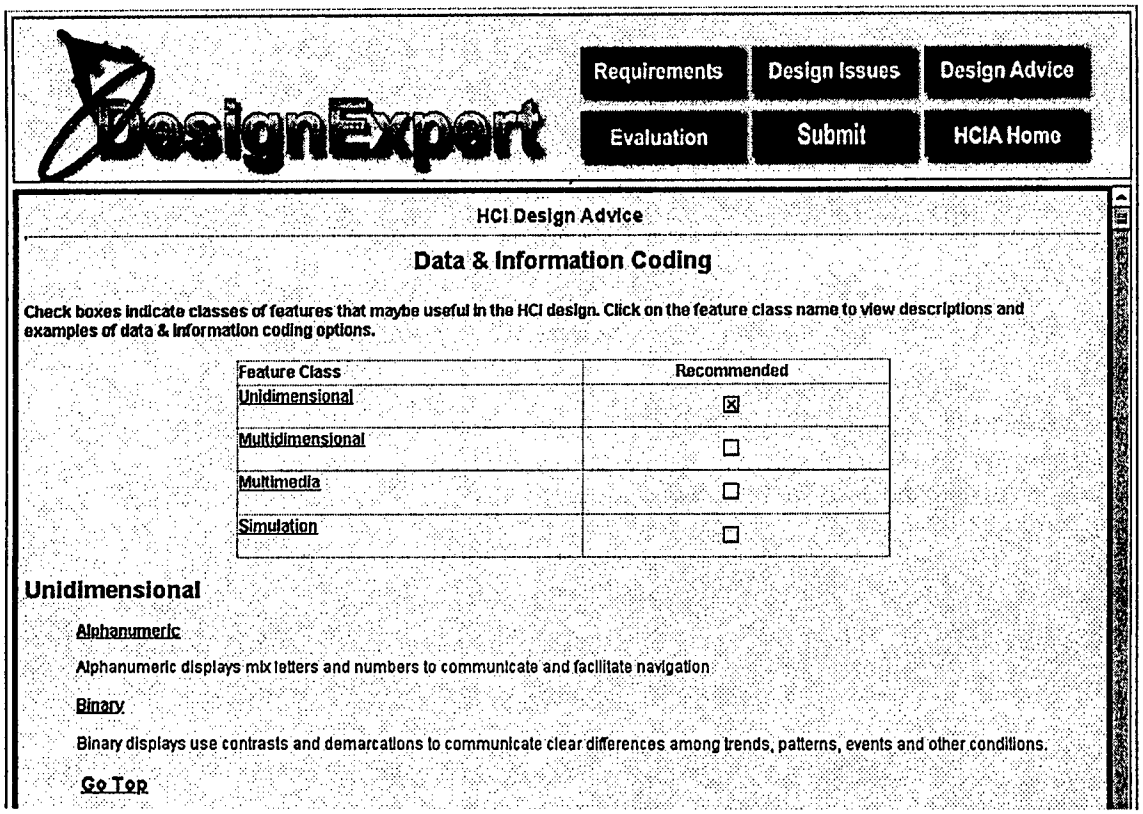


Figure 6.2: Sample HCIA Advisor design advice for data and information coding

The HCIA Advisor presents definitions and an example for each possible feature and a recommendation as to its appropriateness for the system under development. Figure 6.2 shows a sample HCIA Advisor design advice for data and information coding.

To complete the lifecycle support, the HCIA Advisor provides reference information on the critical design evaluation issues and suggests approaches to evaluation of the HCI design, user performance, and system usability. The evaluation screens identify and describe a range of objective and subjective methods for assessing performance and workload (a sample screen is shown in Figure 6.3). Throughout the HCIA Advisor, HTML links connect the user to further information available online.

The HCIA Advisor is designed to be used by analysts and design decision makers, including domain knowledgeable customers (e.g., operational end-users, domain experts, etc.) and software development practitioners (e.g., project managers, analysts, designers, programmers, quality experts, etc.).

The HCIA Advisor can be extended by creating new cases that reflect issues such as the following:

1. Significant changes in interface technology or interaction paradigms.

2. Application of HCIA Advisor templates to domains that are not characterized primarily as control systems or decision support systems (e.g., accounting, data warehousing

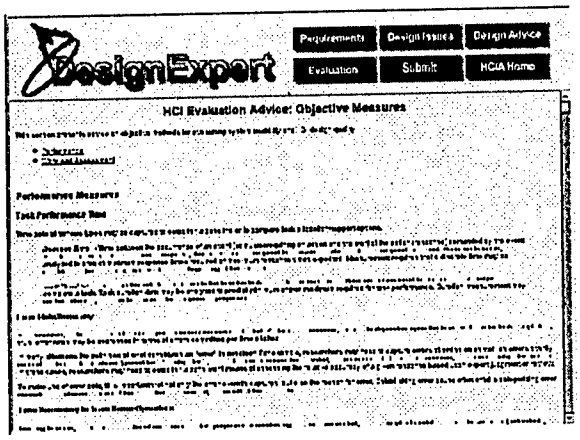


Figure 6.3: Sample HCIA Advisor evaluation advice

applications, etc.).

In each case, the extension of the knowledge base requires either an understanding of the cognitive system engineering principles and related research that support them or sufficient experience and expertise to make the necessary judgments. The changes are simply made to a Microsoft Excel file which, in a simple format, represents the HCIA Advisor knowledge base. This Excel file is then compiled into a format that the HCIA Advisor can access directly.

6.3 The HCIA Critic

As mentioned above, the HCIA Critic provides an assessment and a critique of a particular interface as part of the evaluation phase. The HCIA Critic takes into account the goals of

the system, the HCI-relevant characteristics of the system elicited during the requirements analysis, and the given advice.

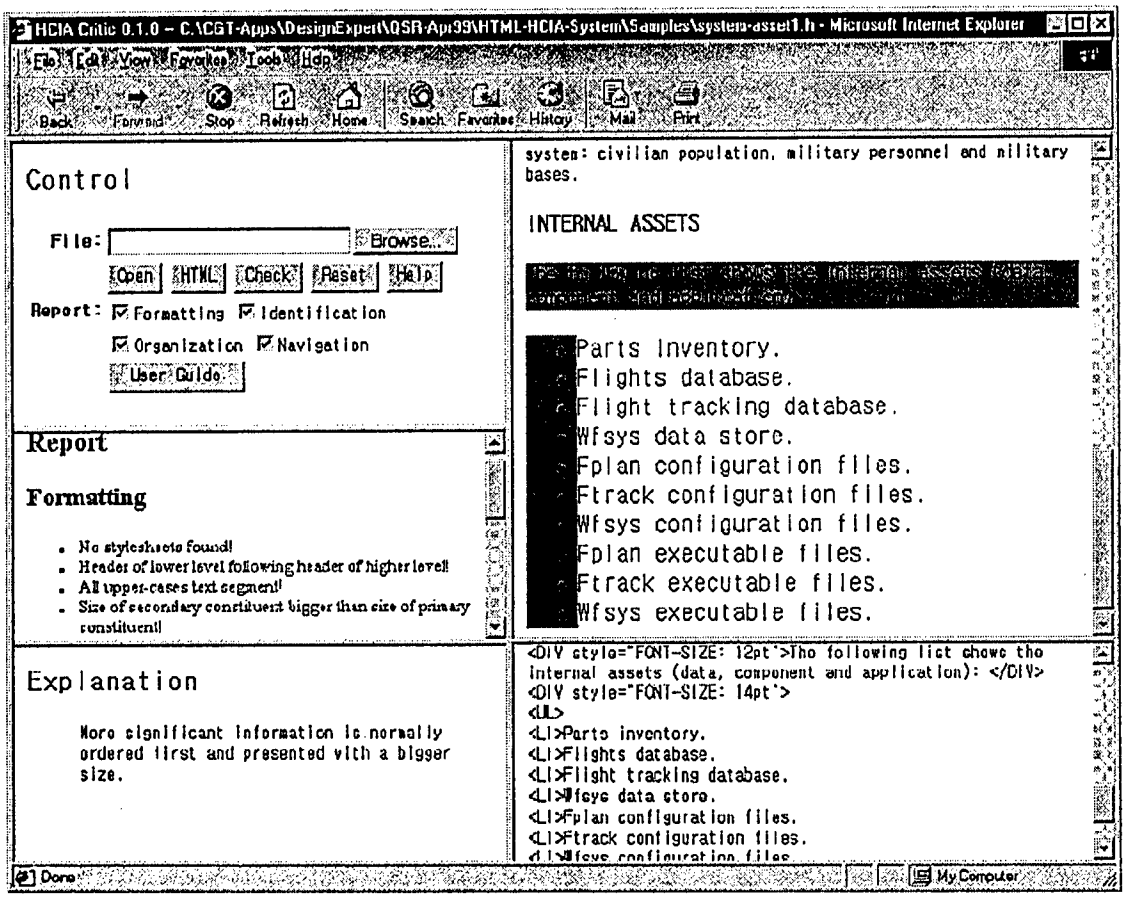


Figure 6.4: Sample HCIA Critic main page

The HCIA Critic focuses on HTML-based interfaces. This choice was made, in consultation with Prof. Scott Overmyer of Drexel, for three reasons:

1. The vast popularity of such interfaces provides abundant material for the validation of the critiquing component.

2. The relatively limited "vocabulary" of HTML-based interfaces (as compared with generic GUI possibilities) makes the assessment and critique more focused and concrete. The critique is performed according to the best publicly available HTML style guidelines.

3. There are currently no tools critiquing the style of HTML-based interfaces using style guidelines.

HCIA Critic checks HTML documents for possible violations of web style guidelines, such as those published by Yale or Ameritech. The application reports the results of the checking

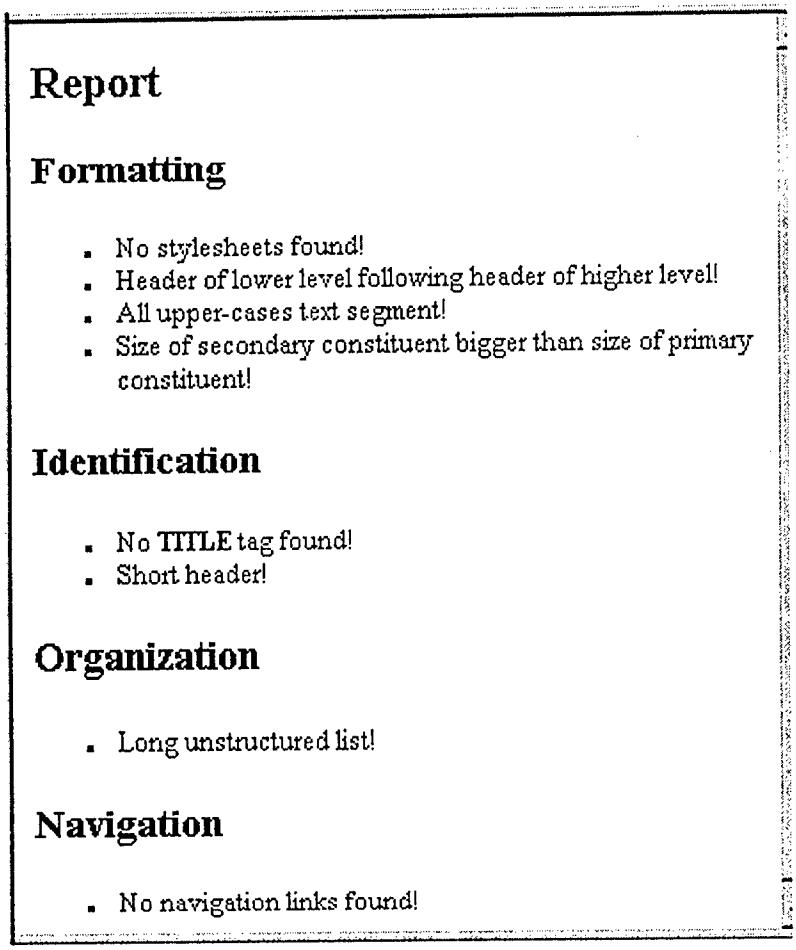


Figure 6.5: Sample HCIA Critic report

using dynamic HTML to highlight the sources of the errors in the rendered documents and in their HTML specifications.

Figure 6.4 shows a sample main page of the HCIA Critic. The Control Frame (at the top left) contains buttons controlling document selection and the content of the other frames. Using this frame, the user selects an html page to load into the HCIA Critic. After loading, the Rendered Document Frame (top right) displays the selected document, rendered by the browser, while its html source code is shown in the frame below it. (in the example in Figure 6.4, the page being analyzed lists internal assets of a site. The relation to the Security Assistant is purely accidental.)

After the page is loaded, the HCIA Critic analyzes the page. The Report Frame (middle left frame) then displays the results of checking the selected document. Figure 6.5 shows the full report for the example. The bulletized items are error messages with hyperlinks which allow the user to get more details about the errors. The activation of a link associated with an error message generates an explanation for this error message in the Explanation Frame (just below the Report frame). In addition, clicking on an error message also highlights the effect and/or the cause of the error in the Rendered Document Frame and/or in the HTML

Frame.

6.4 HCIA Critic: Support for Yale's Web Requirements and Guidelines

This section describes a list of Web requirements and guidelines based on the Yale Style Guide (Lynch, P. and Horton, S. (1998) **Yale C/AIM Web Style Guide**. Center for Advanced Instructional Media at Yale University.
Available at: http://info.med.yale.edu/caim/manual/contents.html) that are supported to some extend in HCIA Critic.

Formatting.

- Avoid table borders altogether, use spacing and indentation instead.
- Use HTML heading tags but not font and styles in *headers* – otherwise search engines will have difficulties
- Avoid all uppercase headlines
- Don't capitalize every word (legibility depends on the tops of words)
- Use proven publishing standards (like Xerox Publishing Standards):
  - Capitalize initial letters in
    - document titles
    - other web sites
    - titles of documents referred to within the text
    - proper names, product and trade names
  - Capitalize only first word in
    - subheads
    - references to other headings within the text
    - figure titles
    - lists
- Use as few heading styles as possible (2)
- Avoid too long lines of text (limit 8cm/3in or 40-60 chars by line). It is approx ½ of normal width of a Web page, allows people to read without moving their head
- Avoid fonts not supported by Windows or MacOS (non-supported fonts will be substituted with unpredictable effect)
- Avoid the combination of <TABLE> and <FONT> tags because it could be unpredictable
- Use "invisible" tables for layout (non-intended use of tables)
- Avoid excessive markup: too many links or too many styles of typeface
- Your link colors should closely match your text color, avoid screaming colors
- Avoid special characters (not supported by standard HTML) and auto hyphens
- Avoid sentences organized around a link phrase such as "click here for more information"
- Remember that for reading links are a distraction, use carefully in text
- Group all minor, illustrative, parenthetic, or footnote links at the bottom of the document where they are available but not distracting

Organization.

- Avoid unnecessary scrolling – it is bad in general because the reader loses context and gets disoriented
- A web page, even meant for down loading, should not be longer than 2-3 printed pages
- Users prefer few very dense screens to many layers of simplified menus
- For a long down loadable text, have it on a link as a file. In that file, always include the url of the on-line page which points to the file.
- Use modular design for easy update –urls remain contact even if pages change
- Keep short:
  - home page,
  - menu and navigation pages,
  - documents to be browsed and read on-line,
  - pages with very large graphics (Yale guide has a table of recommended sizes for graphics)
- Menus should be short but not too short: recommended minimum 5 to 7 links
- Each page should be self-contained and contain all the essential uniform links, name of the company or the web site
- Home page should have a menu
- Avoid slowly loading graphics, especially at the top of home page: Estimated "frustration threshold" – 10- sec
- Always have "what's new" for big sites

Consistency.

- Try to achieve "invisible interface" by uniformity of your pages
- Maintain uniform:
  - layout grid
  - titles, subtitles
  - footers
  - links

Navigation.

- All the links on home page should be "inward"
- Always have top and bottom (especially on pages that scroll) bar with all major links on every page
- Provide a "home page" link on every page

- Use bottom bars to display location, to orient user in the site context
- Always have a "signature header" on pages where users can come from outside
- Provide a link to site editor for complaints/comments on every page
- Provide a FAQ and "related sites" links, they are always appreciated by visitors
- Provide origin and age of the page
- All links should be *current* (functional)

Bibliography

(1985). *Trusted Computer System Evaluation Criteria.* US Department of Defense.

Andriole, S. J. and Adelman, L. (1995). *Cognitive Systems Engineering for User-Computer Interface Design, Prototyping and Evaluation.* Lawrence Erlbaum, Hillsdale, NJ.

Avizienis, A. and Kelly, J. (1984). Fault tolerance by design diversity. *IEEE Computer*, 17(8).

Barzilay, R., McCullough, D., Rambow, O., DeCristofaro, J., Korelsky, T., and Lavoie, B. (1998). A new approach to expert system explanations. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario.

Davis, A. M. (1994). *Software Requirements: Objects, Functions, and States.* Prentice-Hall, Inc., Upper Saddle River, NJ, 2nd edition.

DoD Instruction 5200.40. *DoD Information Technology Security Certification and Accreditation Process.* DISA.

Ehrhart, L. S., Andriole, S. J., Monsanto, C., Hibberd, B. J., and Flo, R. (1996). Designpro: Expert advice for information interaction design, prototyping, and evaluation. In *Proceedings of 1996 Sixth Annual Dual-Use Technologies Applications Conference*, Syracuse, NY.

Fleischman, E. A. and Quaintance, M. K. (1984). *Taxonomies of Human Performance: The Description of Human Tasks.* Academic Press, New York.

Garlan, D., Monroe, R. T., and Wile, D. (1997). Acme: An architecture description interchange language. In *Proc. CASCON '97.*

Giarratano, J. and Riley (1994a). *Expert Systems: Principles and Programming.* PWS Publishing Company, Boston.

Giarratano, J. and Riley, G. (1994b). *Expert Systems, Principles and Programming.* PWS Publishing Company.

Kittredge, R., Korelsky, T., and Rambow, O. (1991). On the need for domain communication knowledge. *Computational Intelligence*, 7(4).

Lavoie, B. and Rambow, O. (1998). A framework for customizable generation of multi-modal presentations. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, Montréal, Canada. ACL.

Lester, J. C. and Porter, B. W. (1997). Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Computational Linguistics*, 23(1):65–102.

Mann, W. C. and Thompson, S. A. (1987). Rhetorical Structure Theory: A theory of text organization. Technical Report ISI/RS-87-190, ISI.

McKeown, K. (1985). *Text Generation.* Cambridge University Press, Cambridge.

Meister, D. (1981). *Behavioral Research and Government Policy: Civilian and Military R & D.* Pergamon Press, New York.

Moore, J. (1994). *Participating in Explanatory Dialogues.* MIT Press.

National Institute of Standards and Technology et al. (1996). Common criteria for information technology security evaluation. http://csrc.nist.gov/nistpubs/cc.

Nieuwenhuis, L. (1990). Static allocation of process replicas in fault tolerant computing systems. In *IEEE Symp. Fault Tolerant Comp.*

Paris, C., Wick, M., and Thompson, W. (1988). The line of reasoning versus the line of explanation. In *Proceedings of the 1988 AAAI Workshop on Explanation*, pages 4–7.

Paris, C. L. (1988). Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78.

Rasmussen, J., Pejtersen, A.-M., and Goodstein, L. (1990). Taxonomy for cognitive work analysis. Technical Report Riso M-2871, Riso National Laboratory, Roskilde, Denmark.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: Mycin.* American Elsevier, New York.

Siewiorek, D. and Swarz, R. (1982). *The Theory and Practice of Reliable System Design.* Digital Press.

Swartout, W. and Moore, J. (1993). Explanation in second generation expert systems. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 543–585. Springer Verlag.

Swartout, W., Paris, C., and Moore, J. (1991). Design for explainable expert systems. *IEEE Expert*, 6(3):59–64.

Tanner, M. C., Keunecke, A. M., and Chandrasekaran, B. (1993). Explanation using task structure and domain functional models. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 586–613. Springer Verlag.

Webber, F., McEnerney, J., and Kwiat, K. (1998). The DesignExpert approach to developing fault-tolerant and secure systems. In *4th Int'l Conf. on Reliability and Quality in Design.*

Wick, M. R. (1993). Second generation expert system explanation. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 614–640. Springer Verlag.

*U.S. GOVERNMENT PRINTING OFFICE: 2001-710-038-10187

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

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