TOWARDS REFERENCE MODELS FOR REQUIREMENTS TRACEABILITY

Bala Ramesh
Department of Computer Information Systems, Georgia State University
Atlanta, GA 30303, USA
bramesh@gsu.edu

Matthias Jarke
Informatik V, RWTH Aachen
Ahornstr. 55, 52074 Aachen, Germany
jarke@informatik.rwth-aachen.de

May 1999

Requirements traceability is intended to ensure continued alignment between stakeholder requirements and system evolution. To be useful, traces must be organized according to some modeling framework. Indeed, several such frameworks have been proposed, mostly based on theoretical considerations or analysis of other literature. This paper, in contrast, follows an empirical approach. Focus groups and interviews conducted in 26 major software development organizations demonstrate a wide range of traceability practices with distinct low-end and high-end users of traceability. From these observations, reference models comprising the most important kinds of traceability links for various development tasks have been synthesized. The resulting models have been validated in case studies and are incorporated in a number of commercial traceability tools. The discussion of the link types and their usage in practice has implications for the design of next-generation traceability methods and tools.

This work was supported in part by grants from the Office of Naval Research, the European Commission (ESPRIT project 21.903 CREWS), the Deutsche Forschungsgemeinschaft under project Ja445/5-1 (PRIME), and by a cooperation grant on empirical and formal methods for requirements traceability, jointly funded by the German DAAD and the US National Science Foundation.

1. INTRODUCTION

Reference models are prototypical models of some application domain, usually organized according to some underlying basic metamodel. The purpose of reference models is to reduce significantly the task of creating application-specific models and systems : the user selects relevant parts of the reference model, adapts them to the problem at hand, and configures an overall solution from these adapted parts. Since the analysis of a domain can take an enormous effort when started from scratch, the use of reference models has been reported to save up to 80% in development costs for systems in standardized domains (Scheer 1994). Not surprisingly, reference models have become highly successful in many industries, the best-known example being the SAP approach.

Not every domain is sufficiently standardized to allow for a reference model of the final product – the system to be built. Moreover, approaches like the one followed by SAP are not necessarily supportive of change, especially when this change goes beyond the initially covered domain. However, at least experience on how to get to the product should be reused (Dhar and Jarke 1985). This leads to the idea of reference models for capturing the development process itself, not to be confused with prescriptive software process models (Finkelstein et al. 1994).

The process of developing reference models is arduous. The traditional normative computer science approach of imposing such models on developers is long known to have failed in most cases. Reference models are therefore an abstraction of best practice, condensed from numerous case studies over an extended period of time, followed by more case studies to refine and evaluate the proposed reference model. There is nothing provably correct about reference models; they derive their relevance from the slice of practice they cover. Nevertheless, by formalizing a reference model in an appropriate mathematical framework, at least a number of elementary desirable properties can be ensured.

1.1 Why Reference Models for Traceability?

Our interest is the development of reference models for requirements traceability. Summarizing several earlier views and perspectives, Gotel and Finkelstein (1994) define requirements traceability as "*the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases*". The importance of requirements traceability is highlighted by the fact that, for example, the US Department of Defense spends about 4% of its IT costs on traceability – often without getting an adequate value for this money, as traceability in many organizations is haphazard, the standards provide little guidance, and the models and mechanisms vary to a large degree and are often poorly understood. Not surprisingly, the market for requirements traceability tools is booming, even though current tools support only rather simple traceability models and services.

Previous models of requirements traceability focus on different aspects of requirements traceability. Version and configuration management systems focus on the **SOURCE** aspect (i.e., physical artifacts where traceability information is maintained), emphasizing the document management role of traceability (Rose et al. 1991, Conradi and Westfechtel 1998). The source aspect remains important: only trace objects available in *persistent* sources actually constitute long-term traceability, so traceability methods and tools have to make traces persistent reliably and at acceptable cost.

Studies at the management level usually focus on the **STAKEHOLDER** aspect (i.e., agents involved in the management of traceability). A good example is the work of Yu and

Mylopoulos (1994) who use dependencies between stakeholders as a starting point for driving and documenting requirements processes. The roles of stakeholders with respect to objects are emphasized in Gotel's thesis work on contribution structures (Gotel and Finkelstein 1995). Her empirical study shows the practical value of using a rich system of stakeholder roles in documenting the creation of requirements (Gotel and Finkelstein 1997); our longitudinal case study (Ramesh et al. 1997) shows that also the different usage roles are important in designing and implementing traceability systems. Summarizing, the stakeholder aspect becomes more important as the duration and complexity of projects grow.

Despite the importance of considering physical trace management and stakeholder roles, current practice shows that the efficiency and effectiveness of traceability support is largely determined by the system of **OBJECT** types and traceability link types offered. This is not only important for fine-grained change management, but also has repercussions for the two other aspects, i.e. for source management and stakeholder dependencies (Pohl et al. 1997). In this paper, we therefore focus on the analysis of object types and trace link types used in current practice, and organize our reference models around them. However, after presenting the models, we shall discuss their embedding with the stakeholder and source management aspects, including the implications for tool design drawn from our studies.

1.2 Research Overview

Our own efforts to develop a reference model of traceability were driven by experiences with the deployment of REMAP, a model for design rationale and dependency management we developed in the late 1980's (Dhar and Jarke 1988, Ramesh and Dhar 1992). The development of this model was guided by experiments with experienced software engineers, and it has in fact served as a starting point for several commercial tool developments including the KBSA environment by Anderson Consulting and the SLATE tool by TD Technologies. Despite this apparent success, some users of REMAP pointed out that it was too complex for some usage situations, even though each of its features was needed *sometimes*. The informal feedback gained in these discussions was that a *broad variety of traceability strategies* is practiced in industry and the existing models are too simple and/or too rigid to deal with this variety.

In 1992, we therefore embarked on a two-pronged strategy. In the European NATURE project, we began to develop model-driven approaches to process guidance and traceability that would provide the technical means for dealing with a wide spectrum of traceability techniques. As a starting point for this activity, we developed a basic traceability framework that has become known as "the three dimensions of requirements engineering" (Jarke and Pohl 1993, Pohl 1994). In (Pohl 1996), this framework was elaborated by a set of 18 types of traceability links found in the literature, and by the RSM model which synthesizes the objects to be traced from 21 international software engineering standards. In (Dömges and Pohl 1998, Pohl et al. 1999), the resulting process-integrated tool environments are discussed.

In parallel, we began a four-year series of empirical studies with American software development organizations, with the goal of

- capturing the current and desired practice in requirements traceability
- developing and validating reference models which could guide an improved practice
- assessing the impact of good traceability on the software development and maintenance process.

This paper presents the reference models together with their grounding in the empirical studies, and relates the findings to requirements for future traceability mechanisms. Companion papers discuss the organizational factors that guide the selection user organizations make among such models (Ramesh 1998) and the organizational implementation and impact of traceability, based on a two-year case study in one specific organization (Ramesh et al. 1997).

In section 2, we describe the design of the empirical studies, starting from a brief survey of traceability uses reported in the literature. Section 3 presents the first group of results, a simple meta model according to which traceability schemes can be organized. The studies show that there is a clear distinction between low-end and high-end users of traceability representing very simple and very comprehensive traceability practices (explained in detail in Section 2.5.2). Therefore, sections 4 and 5 describe low-end and high-end reference models separately, and indicate ways how to migrate from one to the other. Section 6 contains a fairly detailed example on use of high-end models for supporting a large system-engineering project.

Sections 7 and 8 discuss the implications of our results. In section 7, we define four basic categories of traceability link types and show how they map to the concrete link types in the reference models; by discussing available support for each link type, we relate our results to existing research. In section 8, general conclusions concerning the functionality of traceability tools are drawn; the application of these conclusions is demonstrated by another case study, the development of the Design Rationale tool within the KBSA ADM system of Andersen Consulting. Section 9 summarizes our results.

2. Empirical Studies

In this section, we describe the empirical studies used for defining, validating and applying the reference models.

2.1 Background: Definitions and Uses of Traceability

Requirements traceability has been identified in the literature as a quality factor - a characteristic a system should possess and include as a non-functional requirement (Roetzheim 1991) Many standards governing the development of systems for the U.S. Government (e.g., MIL-STD-2167-A and MIL-STD-498 which replaces it (DoD, 1988)) require the development of requirements traceability documents.

Edwards and Howell (1991) define traceability as a technique used to "provide a relationship between the requirements, the design, and the final implementation of the system". Palmer (1997) states that "traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design and implementation". These relationships allow designers to show that the design meets the requirements and help early recognition of those requirements not satisfied by the design. Palmer states that traceability helps acertain "how and why system development products satisfy stakeholder requirements". Hamilton and Beeby (1991) view traceability as the ability to “discover the history of every feature of a system” so that the impacts of changes in requirements can be identified. Greenspan and McGowan (1978) state that the ability to allow changes to any artifacts -- requirements, specification, implementation-- to be traced throughout the system is an important property of any systems description technique. In short, requirements traceability is a characteristic of a system in which the requirements are clearly linked to their sources and to the artifacts created during the system development life cycle based on these requirements.

Many different stakeholders -- project sponsors, project managers, analysts, designers, maintainers, and end users -- are involved in the system development life cycle. The traceability needs of these stakeholders differ due to differences in their goals and priorities, and many problems of traceability stem from these differences in interest and understanding (Ramesh and Edwards 1993).

Stehle (1990) identifies some criteria from the perspective of *managing* a system development effort. Traceability helps promote a contractor and contracted method of working, demonstrate that each requirement has been satisfied. Further, it helps ensure that each component of the system satisfies a requirement to ward off 'gold-plating' (Wright 1991), i.e. the addition of expensive and unnecessary features to a system.

In *requirements engineering*, a major challenge is the linking of rationales and sources to the requirements. Traceability facilitates communication between those involved in the project to alleviate these problems. Requirements management can be facilitated by capturing information necessary to understand requirements evolution and verification (Fiksel 1992).

During *design*, traceability allows designers and maintainers to keep track of what happens when a change request is implemented before a system is redesigned (Edwards and Howell 1991). Traceability is helpful if it can link designs to justifications, important decisions and assumptions behind them and the contexts in which design solutions are arrived at (Smithers et al. 1991).

Systems evolution requires a better understanding of the requirements which can only be achieved by tracing back to their sources (Pinheiro and Goguen 1996). Traceability provides the ability to cross-reference items in the requirements specifications with items in the design specifications (Dorfman and Thayer 1990). Therefore, with complete traceability, more accurate costs and schedules of changes can be determined rather than depending on the engineer or programmer to know all the areas effected by these changes.

Last not least, *test procedures*, if traceable to requirements or designs, can be modified when errors are discovered (Brown 1987).

2.2 Motivation and Overall Study Design

As a consequence of these different uses and perspectives on traceability, there are wide variations in the format and content of traceability information across different system development efforts. Even though some detailed proposals for traceability frameworks have been made (Gotel and Finkelstein 1995, Pohl 1996), the current literature and the standards do not provide detailed guidelines on what types of information must be captured and used in what contexts. Though a variety of tools used in the industry provide mechanisms to represent various types of linkages between system components, the interpretation of the meanings of such linkages is left to the user. Thus, there is a clear need for the development of reference models, and guidelines for their use in software development activities. A comprehensive scheme, based on such models can help ensure that traceability is maintained through all phases of the systems development process, from the requirements as stated or contracted by the customer, through analysis, design, implementation, testing and operationalization of the product. This step, from the generic link types to their usage in typical development tasks as encoded in a reference model, requires empirical analysis.

As discussed earlier, providing traceability of requirements to their sources and the outputs of the system development process can be along several dimensions. Different stakeholders contribute to the capture and use of traceability information, often with different perspectives. For example, the types of links of interest to an end user may be different from those of interest to a system designer. An end user may be interested in the answer to the following question: what are the system components that are affected by a requirement? A systems designer may, in addition, be interested in the answer to the following: why and how are the components affected by a requirement?

Further, the semantics of a given linkage as viewed by different stakeholders may differ. For instance, consider the following relationship maintained in a traceability table: REQUIREMENT *LINKED-TO* DESIGN object. An end user may view the relationship as a means to identify a design component that is generated by a requirement. A designer may view the same linkage as providing information on the constraint (i.e., requirement) that restricts a design object. The semantics or the meaning attributed to a linkage is guided by the reasoning that the user will be performing with the linkage. While the end user may be interested in just the connectivity information, the designer may be interested in the repercussions of changes in the solutions with the relaxation of the constraint (or redefinition of a requirement).

We conducted several empirical studies that capture the information needs (with respect to traceability) of different stakeholders involved in the software development process. Data collection methods included evaluation of major traceability tools, structured interviews with practitioners, focus groups involving the various stakeholders, and an in-depth case study of traceability practice in one organization. Due to the elaborate preparation required to assemble experienced practitioners in their organizational settings for these empirical studies, data collection spanned a period of over three years. Subsequently, the models have been used and incorporated in a number of prototypical and commercial tools.

Table 1 summarizes the various steps in our data collection and analysis steps and identifies the major outputs of these phases. Each of these steps will be described in the following subsections.

Stage		**Data Sources**	**Outputs**
Pilot Study		Six focus groups Seven verbal protocols Six structured interviews	• Initial version of meta-model • Detailed design for the empirical study
Analysis of current tools		Data on seven commercial traceability tools and three in-house traceability tools - including review of literature, vendor training, interviews with at least two major users of tools	• Capabilities and short-comings of current tools • Understanding of various approaches to traceability support
Main Study	Phase I	Five focus groups of traceability practitioners One group of experts	• Traceability meta-model • Initial coding scheme for development of reference models
	Phase II	23 focus groups of traceability practitioners Evaluation of traceability documents	• Low-end model of traceability • High-end models of traceability
	Phase III	Two groups of experts drawn from different industries	• Comments and informal "validation" of models
Model Use		Use of models in CASE tools • Design rationale models in KBSA ADM • Requirements management models in Tracenet, RECAP • High-end models in SLATE	• Case studies on the use of models • Industrial uptake of results

Table 1: Summary of data collection and analysis

2.3 Pilot Study

The pilot study determined the appropriateness of the various data collection approaches and guided the development of relevant instruments, such as a questionnaire for structured interviews and focus groups. It involved focus groups for idea generation, protocol analysis to study problem-solving behavior, and structured interviews to evaluate the initial results.

A *focus group* is a semi-structured exchange among a small group of people conducted with a clear agenda. This technique is the most commonly used qualitative data collection methodology used in market research (Templeton 1987).

Protocol analysis is commonly used in building expert systems. The thought process of subjects is captured through think-aloud verbal protocols. The protocols are recorded as subjects think through a problem and verbalize their thoughts. The recorded protocols are analyzed to capture knowledge elements and operators that characterize the problem solving behavior. In our context, the knowledge elements provide insights into the relationships or linkages and the operators provide insights into the reasoning performed with the relationships.

Structured interviews were designed to provide further insights into the nature of the traceability links, as well as potential areas for providing support by explicit representation of these relationships. Further, various types of reasoning performed by the stakeholders were identified such that mechanisms can be developed to aid the tasks of design and maintenance based on the nature of the relationships and the perceived areas of potential benefits.

The pilot study involved 58 master's students in information technology at a graduate university. Many subjects had extensive experience in domains such as ship building and aviation maintenance where traceability is widely practiced. The participants had also completed a comprehensive project involving the analysis and design of a segment of a large-scale, real life system.

Six focus groups, consisting of eight to ten subjects each, were conducted. The discussion was moderated to discuss definitions, need for traceability from different stakeholder perspectives, the issues involved in creating a comprehensive traceability practice. Seven subjects participated in the protocol analysis phase of the project. Think aloud protocols of subjects involved in defining traceability links (among requirements, design elements and implementation) in projects they had participated in were collected. Six subjects participated in structured interviews where the results developed from the above data collection sessions were reviewed and elaborated.

Besides the development of initial versions of our meta-model, the primary outcome of the pilot study was the development of a detailed design for the empirical study. Focus groups and structured interviews were identified as the primary methods for data collection during the subsequent phases. The importance of incorporating both contributors and users of traceability information in the study was also highlighted.

No	Area of Business	Stakeholders involved in study	Project Size (cost, number of requirements)	Traceability focus
1	Computer Hardware, System Integration	system engineers project manager quality assurance, system analysts	over 1 million lines	Requirements management, Satisfying requirements, Contract compliance, Project monitoring and control, Rationale, Change control
2	US Government System Development Organization	project sponsors project manager lead system engineers	multi--billion dollar	Rationale, Project monitoring, Satisfying requirements, Resource management
3	US Government System Development Project Management	project sponsor, contractor engineers,	several hundred million dollars	Standards compliance, project tracking and control, resource management,

No	Area of Business	Stakeholders involved in study	Project Size (cost, number of requirements)	Traceability focus
	Organization	system analysts		change control
4	US Government System Development Organization	systems analysts, project sponsors	1400 requirements	requirements evolution, standards compliance
5	US Government System Testing Organization	test engineers, manager, contractor engineers	1300 requirements	satisfaction of requirements, standards compliance, development of appropriate verification procedures
6	System Integration	system engineers, test engineers, maintenance engineers	over 100 million dollars	satisfaction of requirements, inter-component dependencies, development of appropriate verification procedures, rationale
7	Pharmaceuticals	system engineers, maintenance engineers	over 1000 requirements	satisfaction of requirements, rationale, quality assurance, requirements allocation
8	Utility	project manager, system engineers, maintenance engineers	approx. 15 million dollars	satisfying requirements, change control, project tracking and monitoring, compliance with regulations
9	Telecommunications	project manager system engineers, test engineer	6000 requirements	satisfying requirements, quality assurance, project tracking and monitoring
10	Aerospace	project lead engineer, system engineer	several hundred million dollars	requirements allocation, requirements management (evolution), requirements rationale
11	Aerospace	system engineers, maintenance engineer	over 5000 requirements	requirements allocation, design rationale, component dependencies, resource monitoring
12	Electronics	system engineers, project manager, maintenance engineers	multi-million dollars	quality assurance, satisfying requirements, standards compliance, design rationale
13	Defense Contractor, System Integrator	system engineers	over 25 million dollars	requirements allocation, satisfaction of requirements, requirement management (evolution)
14	Electronics	system engineers	600 requirements	satisfaction of requirements
15	Contractor for Financial Services	system engineers	500 requirements	satisfaction of requirements
16	Govt. contractor	system engineers	900 requirements	satisfaction of requirements, contract compliance
17	Govt. project Management organization	project managers, systems analysts	over 100 million dollars	project tracking and control, contracts compliance, standards compliance, resource management
18	Govt. project sponsor	project sponsors project manager system analysts	over 150 million dollars	development of requirements compliance with standards, mission/organizational needs, project costing/estimating
19	Automobile	system engineers,	multi-million	requirements allocation,

No	Area of Business	Stakeholders involved in study	Project Size (cost, number of requirements)	Traceability focus
		maintenance engineers	dollars	satisfaction of requirements, resource monitoring, rationale
20	Electronics	test engineers, project manager, systems analyst	1300 requirements	satisfaction of requirements, development of appropriate verification procedure
21	Govt. contractor	system engineers	400 requirements	development of traceabilitiy documents required by sponsor (requirements-test matrix)
22	Contract Software Developer	system engineers	1700 requirements	development of allocation and test matrices
23	Government Contracting Organization	contract specialists, project manager	over 10,000 requirements	contract compliance, standards compliance, project monitoring and control
24	Contract software Development and Maintenance	Maintenance engineers	800 requirements	requirements satisfaction, test plan development
25	Aerospace	Configuration management personnel, quality assurance, system engineers	8000 requirements	requirements evolution, system /component evolution, requirements satisfaction
26	Telecommunications	system engineers, systems analysts	6000 requirements	development of requirements, requirements rationale, project costing/estimating

Table 2: Organizations involved in focus groups and interviews

2.4 Analysis of Current Tools

As a first step towards developing a clear understanding of the current practice of traceability, a detailed study of the capabilities of major tools that support traceability was undertaken. All leading case tools on the market were included in the study. These include DOORS, Marconi RTM, Teamwork/RQT, RDD-100, SLATE, RTS, and Requirements Tracer. Three CASE tools developed by system engineering organizations for in-house use were also studied. The study included a detailed review of vendor provided literature, participation in vendor provided training (ranging from 1 to five days) on the tools, and interviews with at least two major users of each of the tools. It provided clear understanding of the capabilities of current tools, the areas of traceability supported by each tool, and shortcomings of the different approaches to traceability supported by them, which was very valuable in the design of focus groups and interviews. The details of the tool analysis are not reported here, as they were subsumed by a later survey of traceability tools, conducted by the International Council on Systems Engineering (see www.incose.org/workgrps/tools/tooltax.html) and extended in (Dömges and Pohl 1998).

2.5 Main Study

The main part of the study comprised 30 focus group discussions in 26 organizations. They were conducted in a wide variety of industries, including defense, government, aerospace, hardware development, pharmaceuticals, utility, system integration, electronics, and telecommunications. Table 2 gives an overview of the organizations involved. Whenever feasible, participants representing different user and consumer perspectives were included. On average the focus groups consisted of five participants. As in the pilot study, focus groups were followed up with selected structured interviews.

The participants had experience in several key areas of systems development including project management, software engineering training, requirements management, software testing, system integration, configuration management, procurement, development support, software quality assurance, systems analysis, maintenance, software implementation. The participants had an average of 15.5 years *experience in systems development*.

2.5.1 Phase I

The focus groups were conducted in two phases. In Phase I, building on results of the pilot study, data from five focus groups involving thirty participants were used in developing a traceability meta-model. The meta-model defines the language in which traceability reference models are described. It provides the primitives to represent different traceability linkages among the various objects produced during systems development. It further represents the agents involved in the systems development process as well as the sources in which traceability information is captured. The meta-model is discussed in Section 4. The meta-model was checked for plausibility and terminology by a group of experts drawn from different industries and stakeholder groups.

An important outcome of this study is also the identification of various types of traceability information and the development of an initial scheme for encoding reference models from data collected in subsequent phases.

2.5.2 Phase II

The second phase involved the development of reference models of traceability practice. Data from 23 focus groups were used in the development and classification of traceability links representing current practice and ideal practice.

Issues addressed in the focus groups

Each focus group started with clarifications on the definitions and terms used. This included the identification of the various stakeholders involved in traceability, the system lifecycle phases covered in the project being discussed, the system components to which traceability is intended, and the various views on traceability.

The focus group participants were then asked to discuss their current traceability practice as well as desired traceability practice. To achieve comprehensive understanding of the current and desired practices, participants were asked to elaborate their ideas from two different perspectives:

- *user perspective*: What types of traceability information is needed and is useful at different phases of the system development process? How is this information captured and by whom and at what phase of the system development life cycle? What are the costs associated with the capture of this information?

- *provider perspective*: What types of traceability information does each stakeholder capture? in what phase of the lifecycle and how is it used and by who and when? What are the benefits associated with the capture of this information?

Wherever available, the participants were asked to answer these questions with specific reference to the reports and documents produced in the organization.

The next part of each focus group was devoted to a discussion of the different types of traceability links used in that organization:

1. How are the various traceability relationships identified in the traceability documents?
2. How are the traceability relationships created/defined and prioritized? Who are the stakeholders involved in managing these links?
3. How are the traceability links maintained, reviewed, updated or changed, by whom and during which activities?
4. What are the mechanisms for defining the semantics of these links and maintaining this?
5. What are the uses of each type of traceability link identified?
6. What are the issues that facilitate or impede the implementation of traceability?

Each focus group session lasted between 1.5 - 2.5 hours. All sessions were audio or video taped for further analysis. When this was prohibited by security or confidentiality concerns, a scribe documented the discussions.

During the focus groups, a deeper understanding of the different types of traceability links discussed in the literature was developed. The usage of these link types in practice was gradually encoded in reference models for the major uses of traceability, as discussed below in section 5.

High-end and Low-end Users

It quickly became apparent that the participants in our study could be categorized into two distinct groups with respect to their traceability practice. We refer to them as low-end and high-end traceability users.

Characteristic	**Low-end traceability user**	**High-end traceability user**
Number of organizations in the study	Nine	Seventeen
Number of participants	Fifty-four	Eighty-four
Typical complexity of system	about 1000 requirements	about 10000 requirements
Traceability experience level	zero to two years	five to ten years
User definition of traceability	documents transformation of requirements to design	increases the probability of producing a system that meets all customer requirements and will be easy to maintain
Main application of traceability	requirements decomposition requirements allocation compliance verification change control	full coverage of life cycle, including user and customer; captures discussion issues, decisions and rationale; capturing traces across product and process dimensions

Table 3: Characterization of low-end and high-end usage of requirements traceability

The characteristics of both groups are summarized in Table 3. The representation of the two groups in our empirical study is also discussed in this table. Briefly, low-end users with just a few years of traceability experience see it simply as a mandate from the project sponsors or for compliance with standards. In contrast, experienced high-end users are experiencing traceability as a major opportunity for customer satisfaction and knowledge creation throughout the systems lifecycle. The perceived importance of traceability grows with system complexity. As the following sections will show, the reference models used for low-end and high-end traceability users differ substantially. However, it should be noted that even an organization with predominantly low-end practice may have "mature" traceability practice in some areas and vice-versa.

Development of reference models

The analysis of the data followed standard procedures used in qualitative research that employ focus groups and interviews. The data was analyzed using a form of content analysis (Eisenhardt, 1989) where the data is categorized into concepts. The primary concepts in our study are the various types of traceability links used in current and ideal practice. The results presented in this paper are based on a "qualitative or ethnographic summary" (Morgan, 1988) and rely on direct quotations from the data. As suggested in Morgan (1988), the analysis was done in two phases. During Phase I, a detailed examination of the data from five focus groups provided the initial categories and concepts along which the data was coded. This scheme was developed iteratively so that it was sufficient to explain the data. The scheme from this study was used in the analysis of data from Phase II to identity and refine components of traceability models. Triangulation (Eisenhardt, 1988) across multiple data sources (multiple participants that represent different stakeholder perspectives from each organization) and across data collection methods (focus groups, interviews and review of traceability documents) helped confirm the findings. Each concept represented in the reference models presented in Section 5 was included only when is observed in practice (as reported as current practice in focus group discussions and/or structured interviews and/or represented in traceability documents) and identified by others as a part of desired practice. Each traceability information in the high-end models is in either use or considered desired practice by at least six organizations in our study.

We have developed two sets of models: The low-end models describe the current practice in low-end organizations. The reference models for high-end users in different systems development activities represent the best practices in the industry.

2.5.3 Phase III

The reference models developed from phase II were presented to two groups of experts for comments and feedback. These two groups were composed of experienced traceability users drawn from different industry groups. The two groups provided informal "validation" of the models presented here and further provided suggestions on the implementation of the models in system engineering environments.

2.6 Model Use in Tools

Based on the suggestions received from the experts and our analysis of the current traceability tools, we identified characteristics of system engineering environments for the implementation of the traceability reference models (cf. sections 7 and 8). To develop and adapt reference models, the software information system requires support for specialization and instantiation, applied both to nodes and links.

In order to ensure formal consistency of the proposed models, and to be able to demonstrate them to users and tool vendors, the meta model and the reference models were first encoded in ConceptBase (Jarke et al. 1995), a knowledge-based meta database management system based on the Telos language (Mylopoulos et al. 1990). We shall use ConceptBase screenshots to show the reference models in Sections 4 and 5.

For initial validation, a ConceptBase prototype of an early submodel was included in the concept demonstrator of the Knowledge-Based Software Assistant (KBSA) project. Demonstrations of these prototypes, together with their empirical grounding, led to the adoption of our reference models in several commercial traceability tools:

- The traceability scheme of SLATE, a leading system engineering tool, is architected around earlier versions of the models presented here.
- The design rationale models presented here are the primary components of the Knowledge Based Software Assistant tools Concept Demo and ADM created by Andersen Consulting on behalf of the U.S. Air Force
- The models have been incorporated in the Tracenet tool that supports the capture of traceability across multiple environments
- The models have been tailored and used in a large commercial and a government system engineering organization (Ramesh et al, 1997). The models have also been tailored to support traceability to formal specifications in a model management system (Ramesh 1997).

Though a detailed discussion of the experiences from these implementations is beyond the scope of this paper, they seem to provide strong evidence of usability and scalability of the models in real-life system engineering. We present two case studies in sections 6 and 8.

3. A Simple Meta Model of Requirements Traceability

In the next three sections, we present the reference models resulting from our studies. We assume that our traceability reference models will be implemented in some trace repository (manual or computerized). It is widely accepted that such a repository will comprise at least three layers (for details, see e.g. Bernstein et al. 1999, Constantopoulos et al. 1995, Jarke et al. 1994, Pohl 1996):

- the meta model defining the language in which traceability models can be defined
- a set of reference traceability models which can be customized within the scope defined by the meta model
- a (possibly distributed) database of actual traces, recorded under the chosen models.

We adopt the convention that we denote node metaclasses (e.g., **STAKEHOLDER**) by small bold caps, and their instances (e.g., CUSTOMER) by non-bold small caps. Similarly, link metaclasses (e.g., ***TRACES-TO***) are denoted by bold italics, specific link types by standard italics (e.g., *REFINES*).

The practitioners and focus groups in phase I of the Main Study) confirmed that the most essential aspects of traceability can be captured in the very simple meta model shown in Figure 1 which thus provides the basic language primitives for categorizing and describing traceability models in more detail.

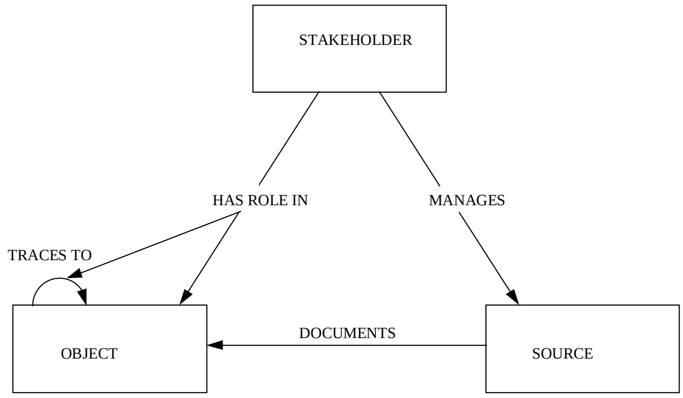


Figure 1: Traceability Meta Model

Each entity and link in the meta-model can be specialized and instantiated to create organization or project specific traceability models. The meta-model can be used to represent the following dimensions of traceability information (cf. Table 4):

1. **What** information is represented - including salient attributes or characteristics of the information?

   In the model, **OBJECTS** represent the inputs and outputs of the system development process. Examples of various types of **OBJECTS** include REQUIREMENTS, ASSUMPTIONS, DESIGNS, SYSTEM COMPONENTS, DECISIONS, RATIONALE, ALTERNATIVES, CRITICAL SUCCESS FACTORS, etc. These represent the major conceptual elements among which traceability is maintained during the various life cycle stages. **OBJECTS** are created by various organizational tasks. Examples of tasks include systems analysis and design activities. This information can be represented as an attribute of **OBJECTS**.
   Traceability across various **OBJECTS** is represented by the ***TRACES-TO*** links. For example, a *DEPENDS-ON* link between two objects (a REQUIREMENT and an ASSUMPTION) can be represented as a specialization of this ***TRACES-TO*** link.

2. **Who** are the **STAKEHOLDERS** that play different roles in the creation, maintenance and use of various **OBJECTS** and traceability links across them?

   In the model, **STAKEHOLDERS** represent the agents involved in the system development and maintenance life cycle activities. Examples of **STAKEHOLDERS** include the project managers, systems analysts, designer etc. These **STAKEHOLDERS** act in different *roles* or capacities in the establishment and use of the various conceptual **OBJECTS** and traceability links.

3. **Where** it is represented **-** in terms of sources that “document" traceability information?

   All **OBJECTS** are documented by **SOURCES**, which may be physical media such as documents or intangible things such as references to people or undocumented policies and procedures. Examples of **SOURCES** include REQUIREMENT SPECIFICATION DOCUMENTS,

MEETING MINUTES, DESIGN DOCUMENTS, MEMORANDA, TELEPHONE CALLS AS WELL AS REFERENCES TO VARIOUS STAKEHOLDERS USING THEIR PHONE NUMBERS, E-MAIL ADDRESS etc. **STAKEHOLDERS** manage the **SOURCES**; i.e., they create, maintain and use them.

4. **How** this information is represented - both by formal and informal means and how it relates to other components of traceability?

   The sources, as mentioned above, can be physical or intangible. Further, they can be represented at different levels of formality. Some sources such as requirements specifications may be text documents, whereas others design documents may be represented in multiple formats such as graphics and text.

5. **Why** a certain conceptual **OBJECT** was created, modified or evolved?

   The rationale behind the creation, modification and evolution of various conceptual **OBJECTS** can be represented as a specialization of the meta-class **OBJECT**. Then, it can then be linked to the conceptual object (using a specialization of the traces-to link). More complex models of rationale, such as the Issue Based Information Systems framework which include issues, alternatives and arguments supporting and opposing them can also be represented as specialization of the **OBJECT-*TRACESTO*-OBJECT** relationship in our model.

6. **When** this information was captured, modified and evolved?

   Relevant temporal information about the any of the entities or links in our model can be represented as their attributes. For example, the frequency or the time / duration at which a requirement or design was created, reviewed, modified or justified by a specific rationale can be represented with this scheme.

DIMENSION	EXAMPLE
What?	Rationale for Design Decisions
Who?	Systems Designer
Where?	In the design documentation library
How?	Using Tool X; Represented as the "Rationale - justifies - Design Decision traceability link
Why?	To facilitate understanding and communication with other designers, maintainer; to avoid rework
When?	At the finalization of the design

Table 4: Traceability Dimensions - An example

The three nodes of the meta model correspond roughly to the three dimensions of requirements engineering proposed by (Pohl 1994) in that they cover the aspects of understanding (objects), agreement (stakeholders), and physical representation (sources). However, note that we are not discussing the requirements process per se, but the creation and usage of traces.

4. Low-End Use of Traceability

A model of the typical low end user's traceability efforts is provided in Figure 2. The names of objects and links presented here are our interpretations (validated by the expert groups) of what was observed in practice. Many organizations simply use traceability tables to link various components of information without explicit identification of the semantics of such relationships. Low-end users created traceability links to model requirement dependencies, allocation of requirements to system components, compliance verification, and change control. For the following, recall that we denote traceability linkages by italic small-caps (*LINKAGES*), while components that they link are indicated with uppercase letters (OBJECTS). For every link in the model an inverse may be defined.

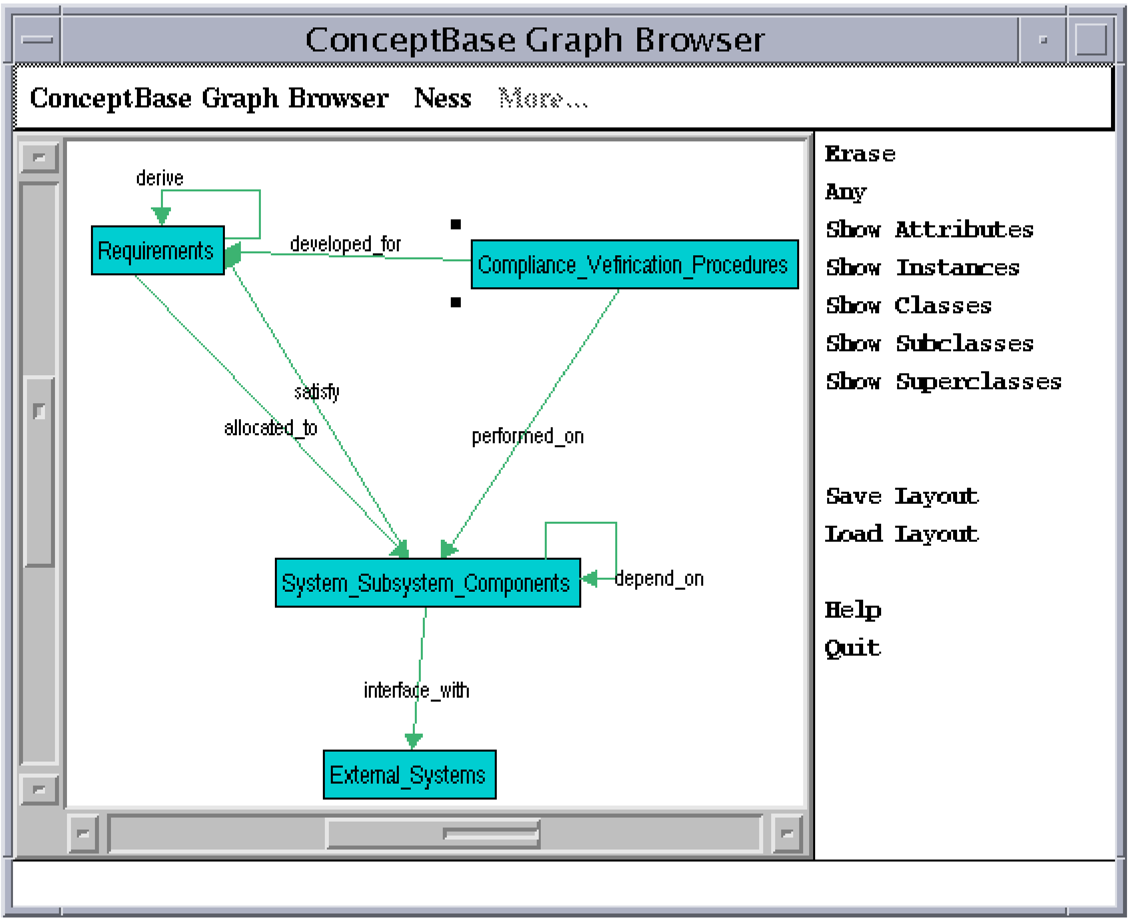


Figure 2: Low End Traceabilty Model

Typical low end users viewed requirements traceability as providing a link from initial REQUIREMENTS to the actual system COMPONENTS that satisfy those requirements. However, first, the higher level requirements must be decomposed to a more refined level. During this recursive process, lower level REQUIREMENTS are *DERIVED* from higher level system REQUIREMENTS. ("*often derived requirements are the problem; you don't know why and how they got there.. want to make sure that every low level requirement is tied to the bigger level specification*") [1]. Typically, this information is captured in a relational data base, and used in the form of a traceability matrix.

[1] This is a direct quote from a subject participating in a focus group or interview. Henceforth, all quotes from a subject will be italicized and included within quotation marks, but no specific reference will be made.

Original and derived REQUIREMENTS are *ALLOCATED TO* system COMPONENTS ("*We want to track which component will take care of what requirement by creating an allocation table*"). An allocation table is the common mechanism used to maintain this information. This simple two way mapping between requirements and system components is usually used also to ensure that there are COMPONENTS in the system that *SATISFY* all REQUIREMENTS ("*when you want to know which part of the system satisfies what requirement, you can go to the allocation table to identify it. Whether the does indeed or not is another matter - for testing to worry about*").

By capturing which components satisfy various requirements and which requirements are mapped to different components, the designer is able to verify that all requirements are addressed by the system.

In the Compliance Verification phase of systems development, low end users use the requirements database, which contains the most current version of the system's validated requirements, to develop the system COMPLIANCE VERIFICATION PROCEDURES (CVPS) such as tests or simulations. CVPS developed for each REQUIREMENT is usually maintained in a REQUIREMENTS-and-TESTS traceability matrix. ("*A matrix showing the tests to requirements is very important to make sure all requirements are adequately taken care of*").

If a change should occur in the requirements then the traceability links could identify the CVPS that must be modified or redeveloped. CVPS are *PERFORMED on* the system components verifying that the component satisfies the requirements. Results of the tests are used to verify that the system works and that it meets all of the requirements. ("*Tests performed indicate whether we indeed achieved success in implementing goals and satisfied the requirements*"). A system COMPONENT may *DEPEND* on others and may also *INTERFACE* with EXTERNAL SYSTEMS. This information is used in evaluating how a requirement is satisfied by a system component.

Low-end users lacked especially in the area of capturing rationale. In requirements management, for example, information concerning requirement issues, how they are resolved, and the rationale for the decisions are seldom captured. Likewise, similar information is absent in the design and implementation phases ("*often we have no idea who made these decisions, and how they impact the rest of the effort. Simply trying to do these at the end of the project or after the fact doesn't work. Often the people who worked on it are gone without a trace of what happened. .. not disciplined enough to document these .. with all the demands on the team*").

5. High-End Use of Traceability

High-end users of traceability employ much richer traceability schemes than low-end users, and also use traceability information in much richer ways. We have, therefore, segregated the high-end model into three parts for clarity: Requirements Management, Design Allocation, and Compliance Verification. In addition, Rationale Management now takes center stage.

5.1 Requirements Management Submodel

Traceability, when implemented correctly, would greatly benefit requirements management facilitating requirements understanding, capture, tracking and verification. The following is a discussion of the Requirements Management model presented in Figure 3.

Systems are built to primarily satisfy ORGANIZATIONAL NEEDS. These needs may either be long term STRATEGIC NEEDS or more immediate OPERATIONAL NEEDS, as denoted by the IS-A hierarchy (shown as thick lines in Figures 3 – 6). These two needs support each other and are often detailed in SCENARIOS *DESCRIBING* the intended use or purpose of the system. ("*We build the system obviously to satisfy some long term or operational goals of our sponsors. In our environment, these are clearly stated in the mission needs statement - MNS - operational requirements document - ORD- two important documents to work within any effort. Obviously, these are at a very high level .. laying out the goals*").

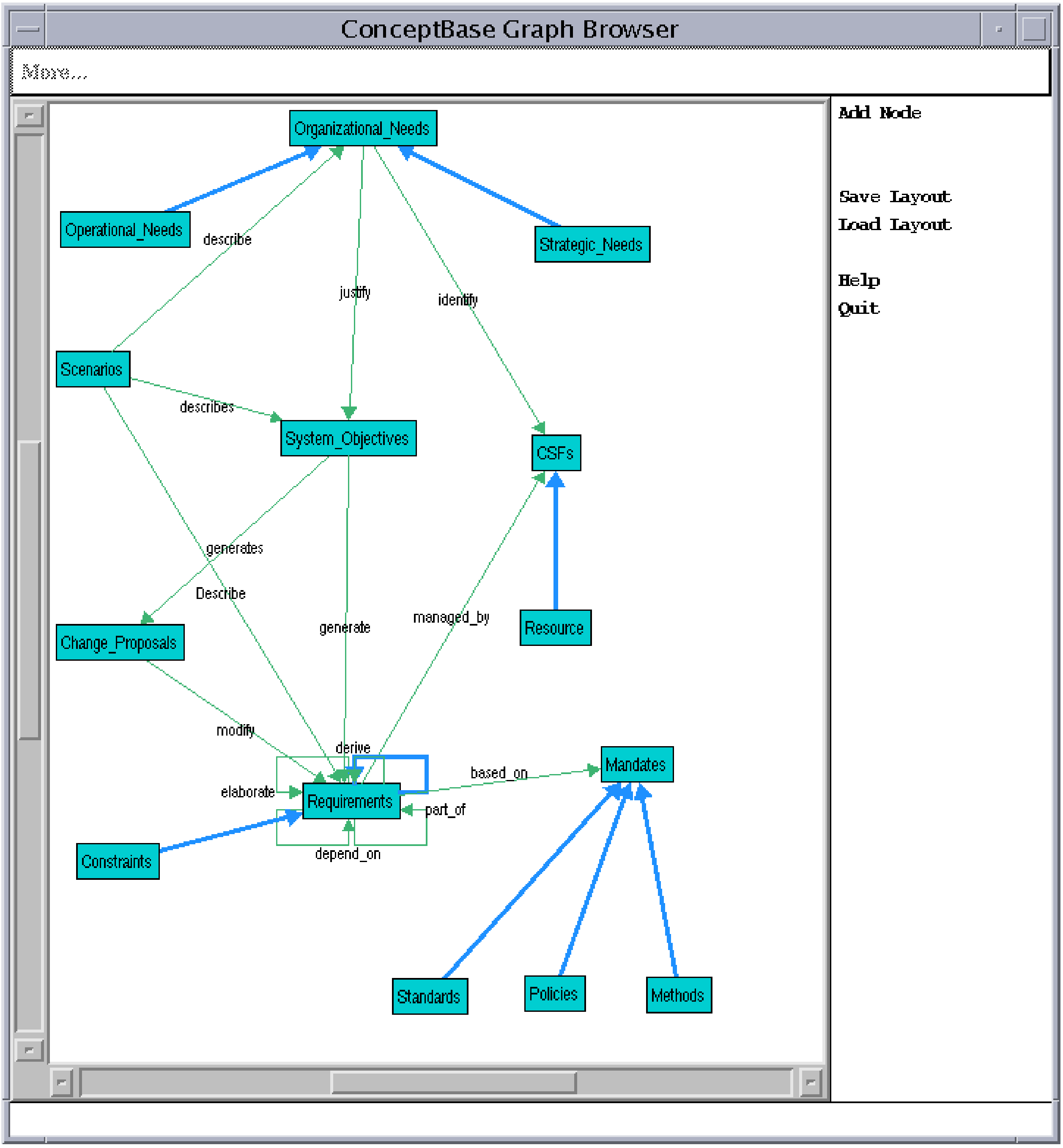


Figure 3: Requirements Management Submodel

The objectives that the system is trying to meet, SYSTEM OBJECTIVES, must be *JUSTIFIED* by ORGANIZATIONAL-NEEDS ("*the system draws its legitimacy from the MNS and*

ORD"). Stakeholders such as customer, program manager, program sponsor, etc. specify the SYSTEM-OBJECTIVES. The REQUIREMENTS for the system are *GENERATED* from these SYSTEM OBJECTIVES (*"The high level goals are after all too vague to be called requirements - for contracting. Specific, detailed requirements have to be developed to proceed further*").

Due to the large number of requirements associated with complex systems, it is important to determine those factors that are critical to the success of the project, and to manage requirements based on how they contribute to these factors. The stakeholders involved in the identification of the ORGANIZATIONAL NEEDS also *IDENTIFY* these CRITICAL SUCCESS FACTORS (CSF) ("*Stakeholders have some issues or values like, no mistakes or system errors, performance is the most important thing, the power requirements are the most critical item in this area*"). RESOURCES such as Cost, Time, Weight, Voltage, etc. are examples of CSFs. ("*A lot of decisions have been made...because of cost and weight, mostly cost.*") ("*It will be a combination of technical and cost trades.*"). REQUIREMENTS for the system are *MANAGED* (budgeted, monitored, tracked etc.) by these CSFs. For example, the mission criticality (a CSF) of requirements could be used in classifying and monitoring requirements. Similarly, CSFs such as weight in an aerospace system or the total cost of development can be used in tracking and monitoring requirements. As part of the negotiation process among stakeholders, many tradeoffs are made in deciding the scope and functionality of the system depending on their impact on CSFs ("*Typically we enter into a complex negotiation process of what's in scope, what's out of scope, what's affordable, what's not affordable. And I believe this is where traceability earns its keep*"). Such decisions determine, among many things, whether requirements identified initially are feasible, cost effective, or desirable.

Not all requirements are equal in significance or criticality; requirements may be traced through the lifecycle at different levels of granularity or detail in order to optimize on resources spent. It may be unnecessary or even undesirable, considering the overhead involved in maintaining traceability, to maintain linkages between every requirement and every output created during the systems design process. The CSFs may be used in deciding when it is essential to trace requirements in detail and when fine grained traceability is less important *("...just try to get as much traceability information for the critical requirements, and not as much for the requirements that may not be as important*"). REQUIREMENTS can be traced throughout the lifecycle to provide stakeholders with a view to understand and evaluate whether the system supports these CSFs ("*...the smaller subset is the driving, high risk set of requirements that merit more visibility in the tracking of the program...we have an attribute of a requirement that we call technical performance measure (TPM). Is it high risk, do we want to put a TPM against this requirement? The TPMs are all programmed, tracking, giving them higher visibility to see how we're doing*"). Further, requirements may also be based on STANDARDS, POLICIES AND PROCEDURES. Maintaining the links between requirements and the sources and the requirements willl help understand and correctly interpret them.

CONSTRAINTS may be treated as a type of REQUIREMENT (denoted by *IS-A* link). Several focus groups stated how constraints become hard requirements because they set limits within which the system is to be developed ("*Systems are designed and decisions made with constraints considered*").

5.1.1 Evolution

One of the biggest challenges in managing large, complex systems is due to the way requirements are constantly evolving and changing. Each focus group noted that in order to accurately reflect this volatility, a requirements traceability scheme should help document and understand the evolution of requirements. Two major sources of changes are: 1) the need to fix a deficiency in the system, identified during, say operations or testing (discussed in detail in

Section 6.4), and 2) changes in ORGANIZATIONAL NEEDS. Various stakeholders modify SYSTEM OBJECTIVES based on changed ORGANIZATIONAL NEEDS, leading to changes in REQUIREMENTS. In projects with very long life cycles, such as military applications, not only is the nature of systems requirements dynamic, but even the ORGANIZATIONAL NEEDS (mission and operational) and the SYSTEM OBJECTIVES keep evolving ("*Systems that were considered successful during the cold war era are no longer considered useful as strategic and mission needs are evolving rapidly*"). It is not possible, however, to abandon all investments in such systems and develop new systems to meet current needs. The lack of comprehensive traceability is a primary reason for the inability to identify components that are linked to various objectives and modify them to meet the current requirements ("*we can't clearly figure out which part of the system we can retain and which part is no longer relevant.. we don't have the traceability details for these systems any more*").

Traceability links are needed among REQUIREMENTS that get specified at different levels of detail as they evolve through the various stages of the development lifecycle. The recursive links among REQUIREMENTS shown in our model are useful in providing a historical record of requirements evolution and in mapping a REQUIREMENT back to its sources, thereby facilitating a complete understanding of where a requirement comes from.

5.1.2 Derived Requirements

Lower level REQUIREMENTS are *DERIVED* from higher level REQUIREMENTS, usually based on assumptions made by stakeholders. Derived requirements need to be tracked carefully as they are likely to be a major source of conflicts and issues, and are subject to changes as assumptions behind them are often unrecorded. Requirements that are derived "*create a collection of requirements at a different level*" of detail. Traceability helps clearly identify derived requirements that "*partially satisfy the originating requirement*" ("*We find things in requirements documents which most people didn't know where they came from. Then we will figure out that someone created these consolidating or clarifying other requirements. We now have a policy to identify - and before that - agree on these derived requirements. There is lot of room for problems when we go from customer given requirements to lower levels based on our interpretations. We try to document these also in as much detail or at least know who is responsible*"). Further, this traceability information is useful in managing the "explosion" in the number of requirements ("*...if you can start with 30 pages of requirements, top level, then you can go to your prime item development specifications and you could consider those requirements derived from those 30 pages, but then you have those second and third generation... derived requirements between those ...you have an explosion of derived requirements and relationships*").

Some REQUIREMENTS are *ELABORATED* by others, providing further explanation or clarification. This added detail assists in understanding the assumptions behind requirements as well as how they are interpreted. ("*Sometimes you add requirements really because you find them unclear. They are refined, and become more specific in detail then in fact are more demanding than the original requirement might have been*").

REQUIREMENTS also *DEPEND-ON* others. For example, a requirement to use a specific software platform may depend on a requirement to use a specific hardware platform. Identifying these dependencies is important, especially when requirements change, so that the impact of the change on the entire system can be determined ("*different people and teams work on different parts of requirements, some even from other parts of the organization or sub-contractors. It is when it is difficult to trace whose part affects which other. After all are part of the same system. We get into major problems down the line when we are unable to figure out these connections*").

Complex requirements are often broken down into their components, identifying simpler requirements that form a *PART-OF* them. These links help in understanding how various pieces of a requirement fit together and are especially important on large, complex systems whose requirements may contain several interdependent parts. (“*we split a requirement to deal with different aspects .. like functionality*”). *IS-A* links between requirements show the parent/child hierarchical relationships. ("*Higher levels of requirements are just abstractions of lower levels*.") ("*Initially its a top-down, hierarchical decomposition starting with the most general goals of the customer, flowing down to a set of systems performance requirements...you now can have children with multiple parents, whereas originally we were probably looking at children with a single parent*."). *IS-A* hierarchies can be used to show the relationship between requirements specified for an entire class of hardware to be used and those for a specific type of hardware component.

5.2 Rationale Sub-Model

The following is a discussion of the Rationale sub-model presented in Figure 4. The specification, elaboration, decomposition, derivation and modification of **OBJECTS** such as REQUIREMENTS, DESIGNS ETC. *GENERATE* (or lead to) ISSUES or CONFLICTS, often due to the differing interpretations, assumptions, interests, viewpoints, experience and objectives of the stakeholders. Information about how decisions are made to resolve these ISSUES must be maintained throughout the system lifecycle to ensure that customer requirements are understood and satisfied ("*Traceability pays for itself by making sure you understand how the requirement is interpreted, how its being implemented, what were the issues that were considered in the design to satisfy that*"). These DECISIONS, in turn, may *AFFECT* REQUIREMENTS.

Various ALTERNATIVES that address the resolution of ISSUES are considered (“*We want everyone to know what the options are, why we are picking one versus another. They can immediately see if these decisions will affect their parts and this feedback is important in making the right decisions. Getting a resolution through this process is also helpful in bringing everyone on board and on the same sheet of music*”). ARGUMENTS for and against each ALTERNATIVE may be proposed.The DECISION to *SELECT* one or more ALTERNATIVES is often *INFLUENCED* by the CSFs (“*All parties understand the various possibilities and why one looks better than others in terms of what is critical for the project*”).

A common observation made by most focus groups is that seldom such detailed information on all the ALTERNATIVES considered, the ARGUMENTS supporting and opposing these alternatives and the assumptions behind this decisions is explicitly recorded. Most often only the chosen alternative is well documented and the discarded ones are not recorded. With evolving requirements, availability of such information may help avoid expensive rework ("*We debated that for two weeks now. Why was it thrown out years ago when we got one camp arguing to put it back in and another to keep it out. And if we could have gone back and assessed why we made that decision two years ago it would have saved us a lot of time*").

Often, the capture of rationale at the detail described above is impractical due to overhead involved in capture and due to the lack of tools to facilitate it. However, simple descriptions of rationale on which REQUIREMENTS OR DESIGNS are based may be recorded along with the assumptions behind them. Further, ASSUMPTIONS underlying the various components of the deliberation are also recorded (“*We record the rationale as simple notes.. and add specifically any assumptions we have made. It is useful to keep track of these assumptions as clearly as possible for these are the most likely candidates for changed requirements*”). We return to this issue in section 8.5.

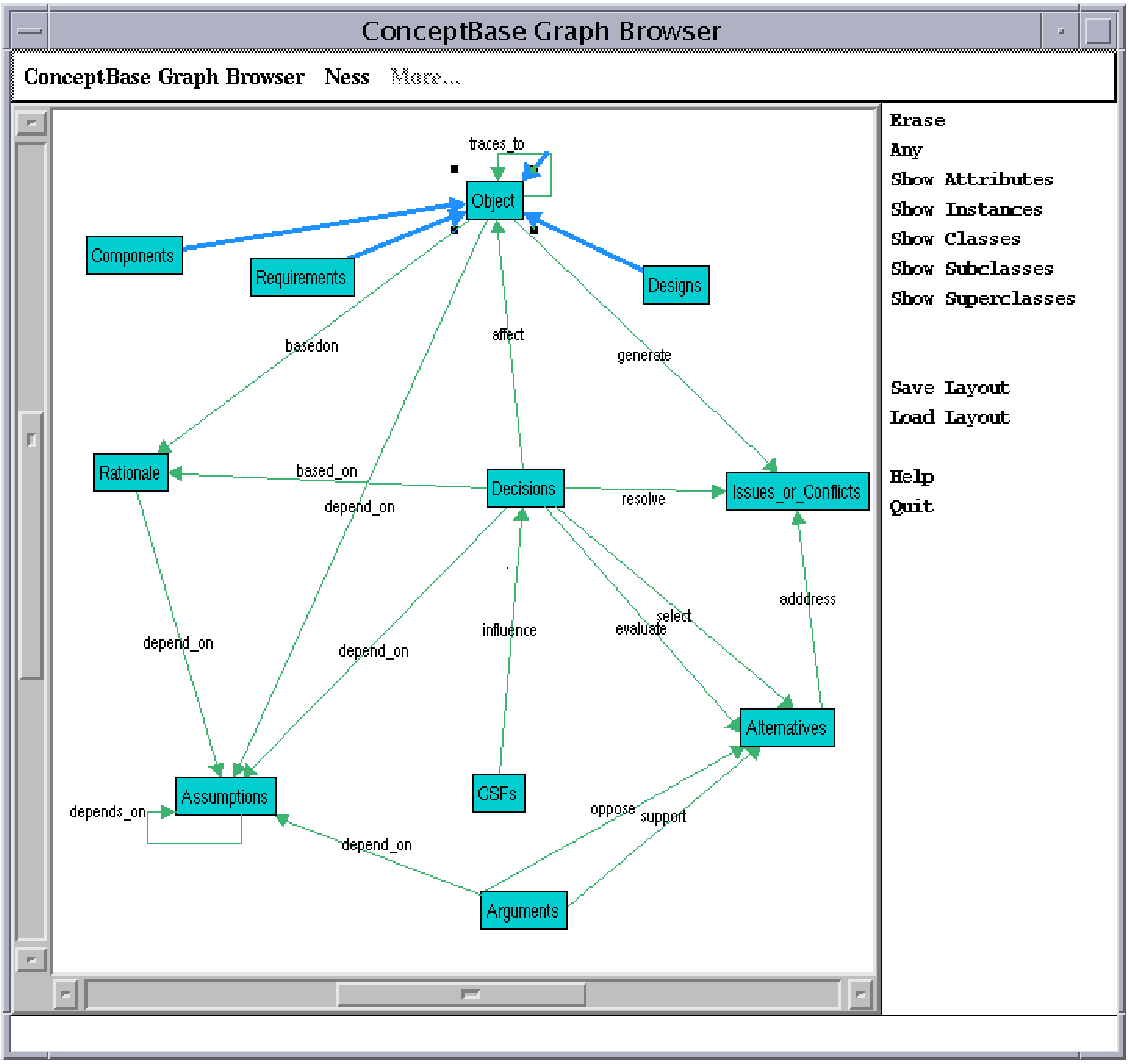


Figure 4: Rationale Submodel

Over the lifecycle of large projects, requirements, assumptions, etc. change. Further, such projects also experience personnel turnover. If detailed rationale is not maintained, these changes lead to a lot of rework ("*We already looked at that tradeoff, that approach, but the person who did it - the analysis - is leaving and the person that reviewed it is gone, so we churn the whole thing again*". "*This is a lengthy look at all these issues with a lot of people that contribute to, touch, change, add, use this information.*").

During the systems development process different stakeholders often bring different interpretations of requirements and their origins. For example, even when two different stakeholders review the same origins of a REQUIREMENT (say, a standard) they may interpret it differently. Often the difference in interpretation is the cause of conflicts between stakeholders, even in a cooperative setting. ("*You can look at a given set of requirements and they might be meaningless if you don't really know what the customer needs because you can interpret them wrong, or at least two or three different ways.*") .

Another reason to have detailed rationale is to provide accountability. Information about how the requirements and design have evolved, what changes have been made, why and how they were made, and the status of their development is extremely helpful to those stakeholders that were not involved in creating the requirement ("*If you don't have traceability to the rationale, it is impossible to understand many low level requirements.. They may have made sense to the person who evaluated the options and not for others*").

5.3 Design/Allocation

We use the term design to refer to any activity that creates artifacts, including implementation. In Figure 5, REQUIREMENTS *DRIVE* DESIGN, that are often *BASED ON* MANDATES such as STANDARDS OR POLICIES OR METHODS that govern the system development activity ("*we have to follow military standards and policies to make sure we are compliant*").

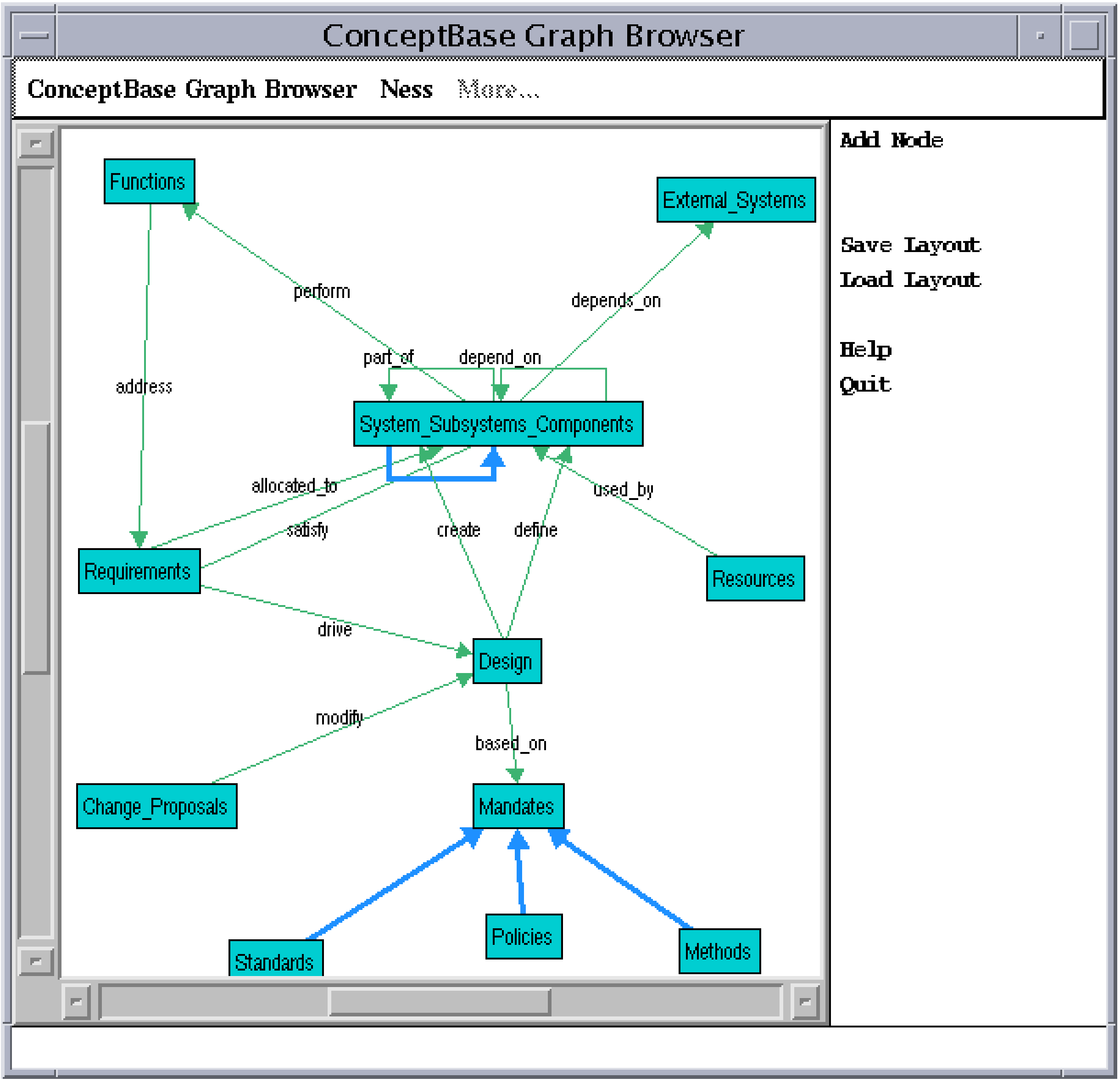


Figure 5: Design Allocation Submodel

SYSTEM/SUBSYSTEM/COMPONENT (hereafter referred to as COMPONENTS) are the building blocks of the system. They are *DEFINED or CREATED* by the DESIGN process. REQUIREMENTS are *ALLOCATED* to COMPONENTS that are supposed to satisfy them. The components could be a piece of hardware, software, humanware, etc. ("*At the allocation step you're going to allocate to people, to processors, to memory, to hardware, etc.*").

COMPONENTS *DEPEND ON* other COMPONENTS, in that the functionality and performance of one may depend on another. When the interfaces between different COMPONENTS are explicitly identified (for example, in interface specifications), maintenance of dependency information is easier. The difficulty in the identification of indirect dependencies among COMPONENTS makes it important to record them when they are indeed discovered. For instance, an operating system on one component may affect the choice of hardware on another, even though there may be no direct interfaces between them. When a component is created or modified, this traceability information is helpful in determining the impact on the entire system. ("*This information is hard to figure out and people who know about how your work on a piece affects some seemingly unrelated piece don't share this information.. leads to lots of rework*").

The composition of COMPONENTS is identified using the *PART-OF* links. This information allows the designers to identify inter-component dependencies ("*As I change a part or break it into sub-parts, I would like to have some sort of traceability or mesh structure which gives some sort of relationships...I would like to know what is the characteristic of each component*"). The ***IS-A*** hierarchy specifies that components are organized at different levels; from system to subsystem to item and to unit ("*we want to keep track common details for same types of components*").

RESOURCES (such as money, weight, personnel, power etc.) are *USED-BY* COMPONENTS. As discussed in Section 5.1, when these resources are CSFs, it is important to track their allocation, distribution and utilization during design and implementation ("*At the allocation level there needs to be parameters that associate budgets of resources that are going to be consumed ... and then you're going to have to test to see that you're somehow compatible with, complying with and not grossly out of bounds with your budgeting*.") ("*I want traceability to give me some level of assurance of particular steps in the design process that I'm meeting or likely to meet system goals or to help me to allocate resources or capabilities in such a way that I maximize the likelihood of meeting system goals*."). ("*If cost happens to be the major system driver, then what I am really interested in is keeping track of actual or supposed cost of the components. If, on the other hand, memory space is the primary driver, they might not really pay as much attention to cost, as how my decisions impact the allocation of memory space*.")

Several focus groups mentioned that it was important to identify the FUNCTIONS *PERFORMED BY* COMPONENTS. These FUNCTIONS are typically traced to the functional REQUIREMENTS explicitly identified in requirements documents ("*we have to keep the functionals separate. We want to make sure each of these is addressed in some way by the different pieces of the system*") ("*Another use of traceability in design is allocating functions in such a way that you're sure that they support each other and don't interfere with each other*"). A COMPONENT may partially *SATISFY* REQUIREMENTS, and as the design progresses, all REQUIREMENTS must be fully satisfied. The difficulty in explicitly creating traceability links between non-functional requirements and system components was also highlighted by several groups ("*with non-functionals, it is a problem. How are we going to say which part of the system affects how much of a performance requirement to give 20 millisecond response*").

The COMPONENTS *DEPEND ON* EXTERNAL SYSTEMS in that they often interface with them. Direct interfaces are easier to recognize and represent in a traceability framework than indirect dependencies ("*a particular product line is going to be brought on board a ship and its going to have to plug into an already existing system...the things they need to know is what is the functionality that the system needs to have, what is the demand on these services that they would have and the range of the demand .. especially with respect to the system we are plugging into*").

Our discussion in Section 5.2 on the capture of rationale applies also to the capture of rationale behind the development of DESIGNS and COMPONENTS. ("Traceability would come in handy for determining the reasons for your design.") ("That's the design knowledge capture that we were supposed to maintain. What is basically here is the design the way it is and here's the decisions, the thought that went into that decision. And then when you changed them, here's how we changed them").

5.4 Compliance Verification Sub-Model

A variety of CVPs (PROTOTYPING, SIMULATION, TESTING and INSPECTION shown by the *IS-A* links) are developed to ensure that each of the REQUIREMENTS is adequately addressed ("*We write the test specifications by taking apart the SRS and Component Specifications*." "*You've got to be able to specify what you're going to test taking Standards into account*"). The ability to develop CVPs is a critical factor in deciding whether a REQUIREMENT is considered as such ("*If something is untestable or we elect not to test, then it becomes no longer a requirement or at least we all agree that it is unverifiable*").

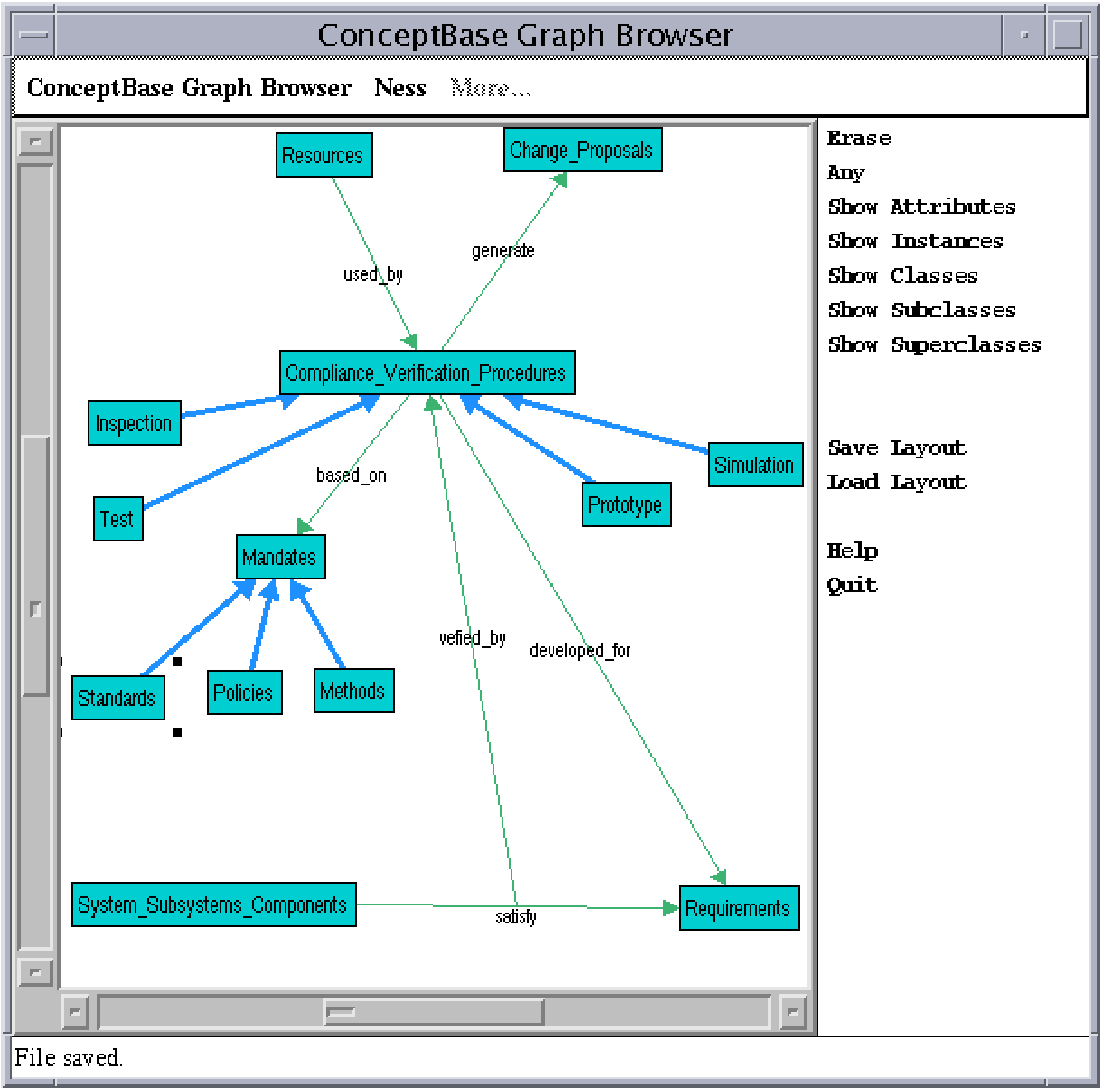


Figure 6: Compliance Verification Submodel

The development of CVPs is also governed by their *USE* of RESOURCES (such as personnel, money etc.). The utilization, assignment and availability of RESOURCES often determines whether one can even perform a compliance test, or if the cost-benefit analysis of performing it justifies it. The tradeoff between the resources allocated to development and CVPs is often a source of ISSUES OR CONFLICTS that must be resolved ("*How are we going to find that this how far we should do go in testing. It is a matter of trade-off, for discussion*").

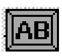
MANDATES such as STANDARDS OR POLICIES OR METHODS are commonly the basis of CVPs and determine which CVPs are required and how they are to be performed. Often, REQUIREMENTS are written such that they comply with these ("*Before we decide on a testing strategy, we want to go to the same standard which the requirement is trying to meet*"). It is important that the CVPs for a requirement use the same interpretation of the standards used while developing requirements. In the absence of traceability between the REQUIREMENTS and the STANDARDS on which it is based, it is difficult to develop CVPs ("*There is a lack of a way to firmly assure that the tests you are generating really reflect the same thing as the standard used in writing the requirement*")

CVPs produce results that either *verify* how the COMPONENTS satisfy REQUIREMENTS or help *GENERATE* CHANGE PROPOSALS for REQUIREMENTS or DESIGN or IMPLEMENTATION. They are thus used to certify completeness and correctness of the system and identify changes that may be necessary to meet the objectives ("*As the design or requirement is changed based on tests, my current system continues to support my objectives.*").

6. Implementing High End Traceability Models: A Case Study

In this section, we illustrate an advanced usage of the models presented in Section 5 with a case study adopted from a system for a U.S. Government organization.

The case study is based on the combat systems section of a new ship (refereed to as ECS hereafter)[2]. The discussion highlights the implementation of traceability at various stages of systems development. We focus on the development of a passive sonar for the ECS, and describe how traceability, using the high-end models described in Section 5, can be implemented in this project. This implementation is described using scenarios at various stages of the development life cycle. The objective of our discussion is to illustrate implementability and tailorability of the models. In this section, the Requirements Management, Design allocation and rationale management are used for this purpose.

The implementation of the models has been using a popular commercial CASE tool, viz., System Level Automation Tool for Engineers (SLATE)[3]. SLATE's traceability scheme is represented using links  among abstraction blocks  (which are building blocks used to represent physical and conceptual objects). User defined subtype of the *TRACE LINK* object can be used to represent specific types of links. The link type is identified within paranthesis with every link. In SLATE, an object that helps define another object is termed a defining object . Objects that comply other objects are called complying objects . For example, a traceability link from a system objective to an organizational need will identify the mission need as defining object. Also, a traceability link from a system objective to a requirement will identify the requirement as a complying object.

[2] The discussion has been "sanitized" to protect proprietary or confidential data.

[3] Trademark of TD Technologies Inc.

6.1 Requirements Management

The traceability scheme used for requirements management is adopted from the high-end requirements management model, and implemented in SLATE as shown in Figure 7. Specifically, it shows the linkages among requirements, mission needs, system objectives, and critical success factors. It also includes components of the rationale model to represent the rationale behind requirements.

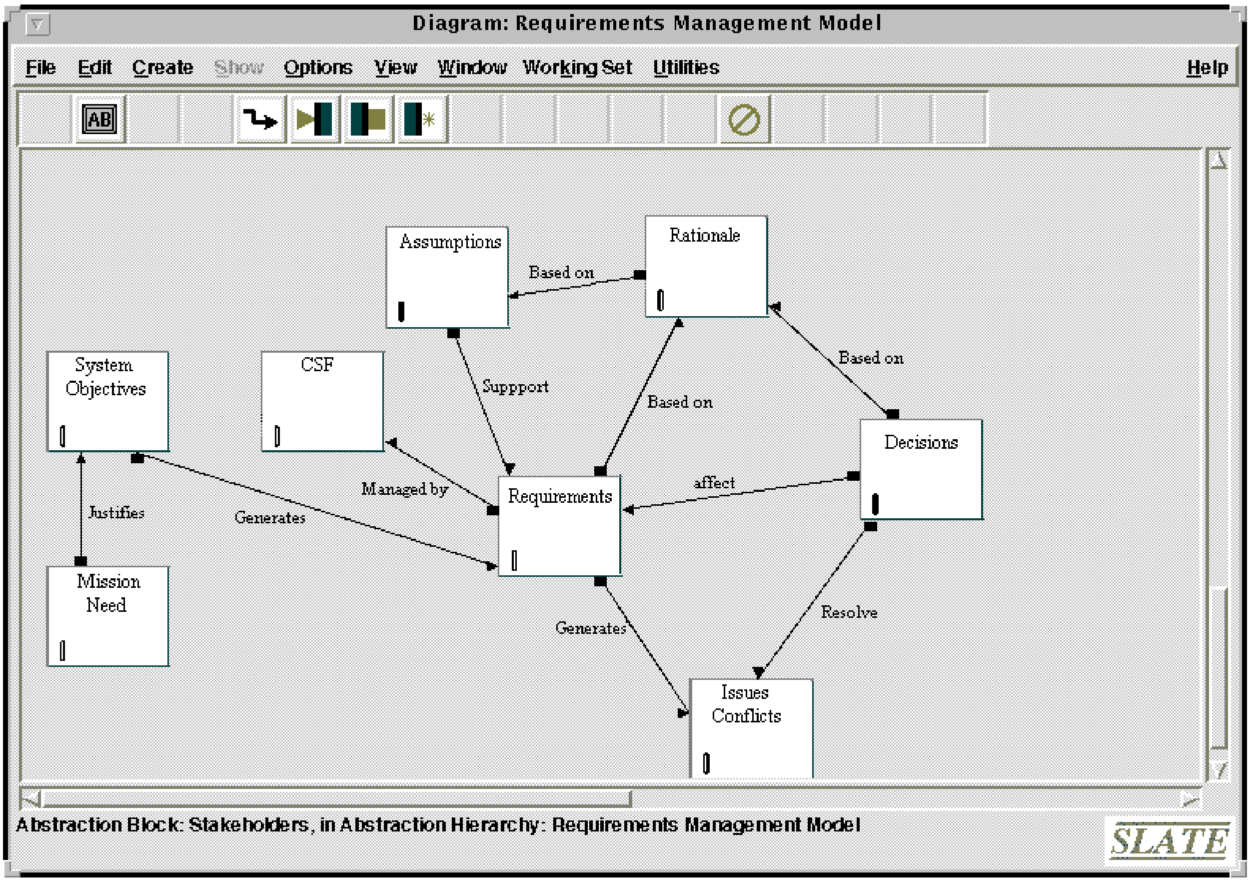


Figure 7: Requirements Management Model

Mission Needs. According to this traceability model, a first step in a system development problem is defining the MISSION NEEDS that *JUSTIFY* the SYSTEM OBJECTIVES. The MISSION NEEDS are provided in a document called the Mission Needs Statement (MNS). This document, a segment of which is shown in Figure 8, resides as a living document in the SLATE database. The MNS document contains the “big picture” requirements for a Surface Combatant ship, the subject of the ECS. It provides general guidelines on the mission, capabilities, design and architecture constraints, and personnel constraints. It also provides some operational constraints.

Each mission needs identified in the MNS is represented in SLATE as an abstraction block. For example, Figure 9 (line 3) shows that a mission need is represented by the abstraction block no. 7, which is also specified as a defining object. In contrast, line 5 identifies a complying object, viz. the system overview. This, in fact, represents the SYSTEM OBJECTIVES (corresponding to the ORGANIZATIONAL NEEDS in figure 3.

The tracing of relevant parts of SYSTEM OBJECTIVES with statements in the MNS is shown in figure 7 using the JUSTIFIES link. The project sponsors involved in the definition of

SYSTEM OBJECTIVES create these linkages to justify each objective based on their relation to higher-level mission needs, i.e. provide “backward traceability” to the origins of requirements. During discussions about the rationale for different functionalities of the system, this traceability will be valuable. For example, the systems analyst can use this information to initially get the “big picture” of what the customer wants the system to do and why.

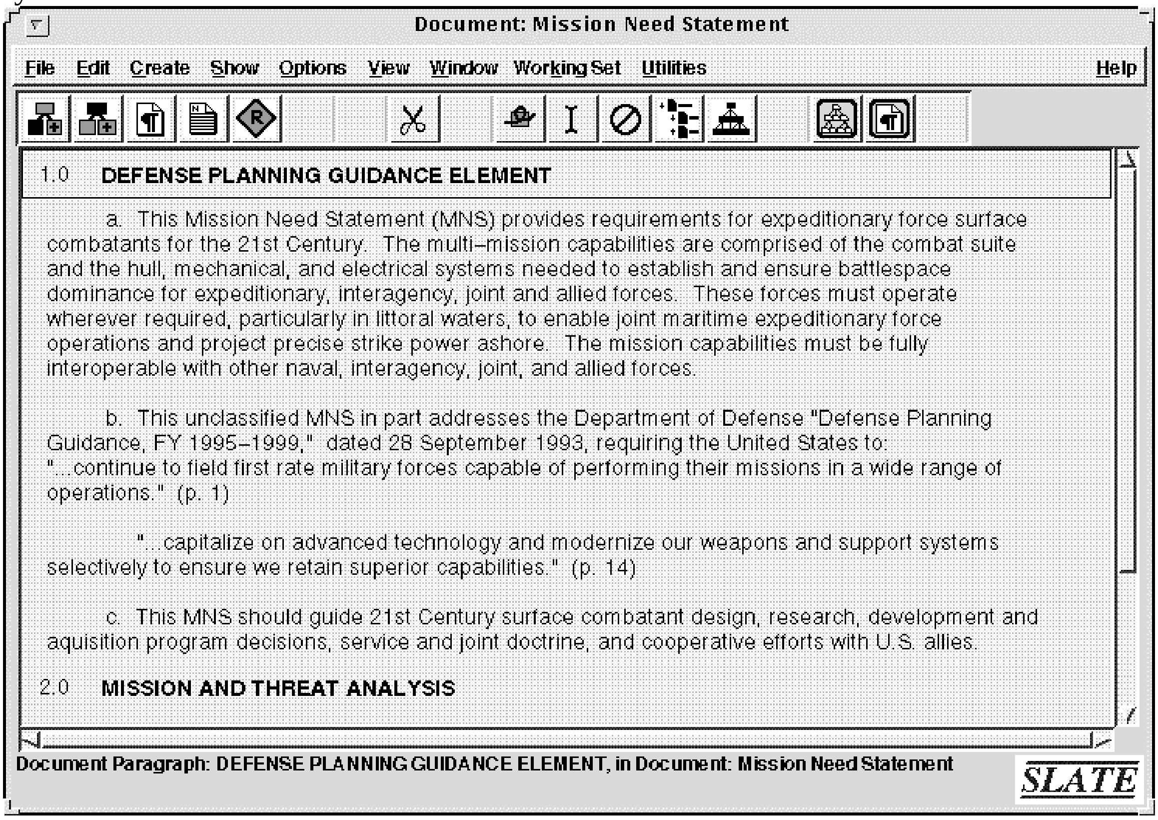


Figure 8: MNS Document

Requirement Identification. The next step in the development is the identification of REQUIREMENTS. As shown in Figure 7, REQUIREMENTS are *GENERATED* from SYSTEM OBJECTIVES. The original requirements document for the ECS that includes requirements for the Passive Sonar Requirements is maintained as a living document in SLATE. A SLATE utility called Requirements Identifier parses documents to identify statements that meet user defined criteria for identifying requirements using keywords. In our case study, in compliance with project standards, sentences that contain the word "shall" are identified as distinct requirements. Parsed text subsequently becomes REQUIREMENTS OBJECTS. In SLATE terminology, these REQUIREMENTS OBJECTS *COMPLY* with the SYSTEM OBJECTIVES. Figure 10 shows System Objective (2.0 System Overview) with several complying requirements (Requirements 3-2 through 3-5).

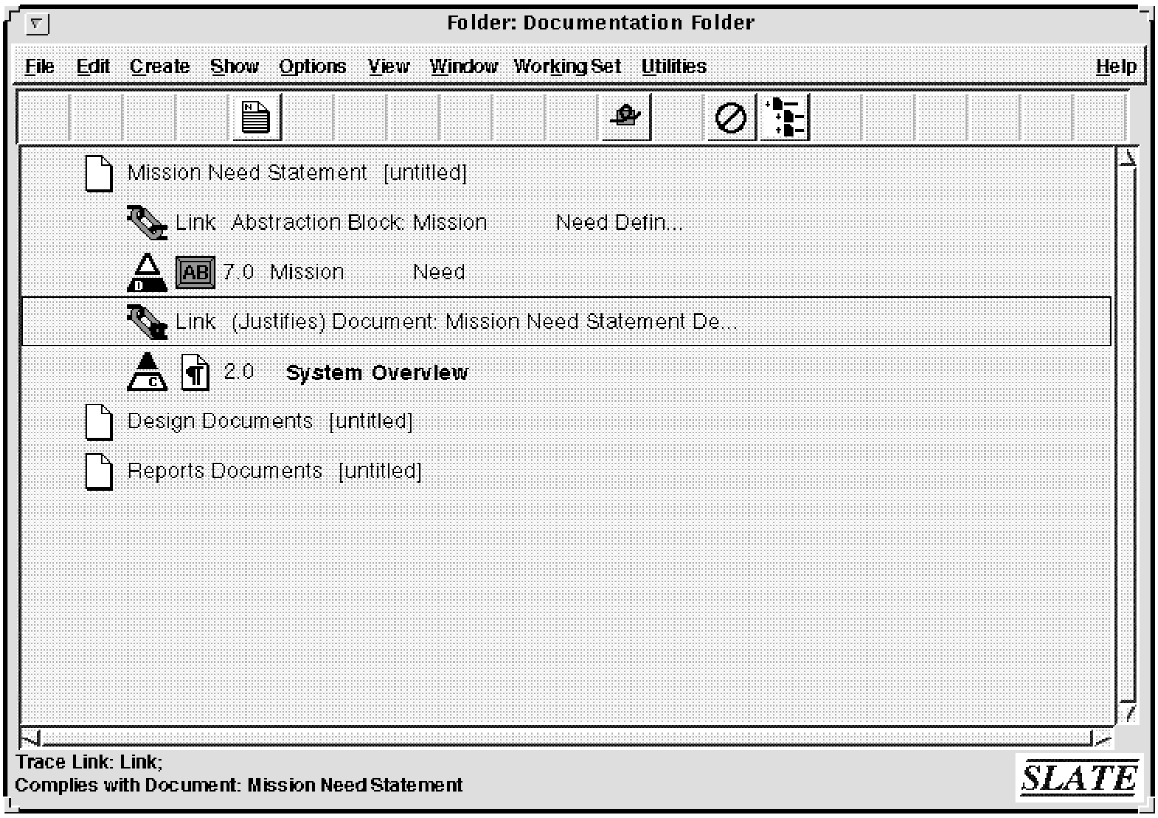


Figure 9 Traceability Links

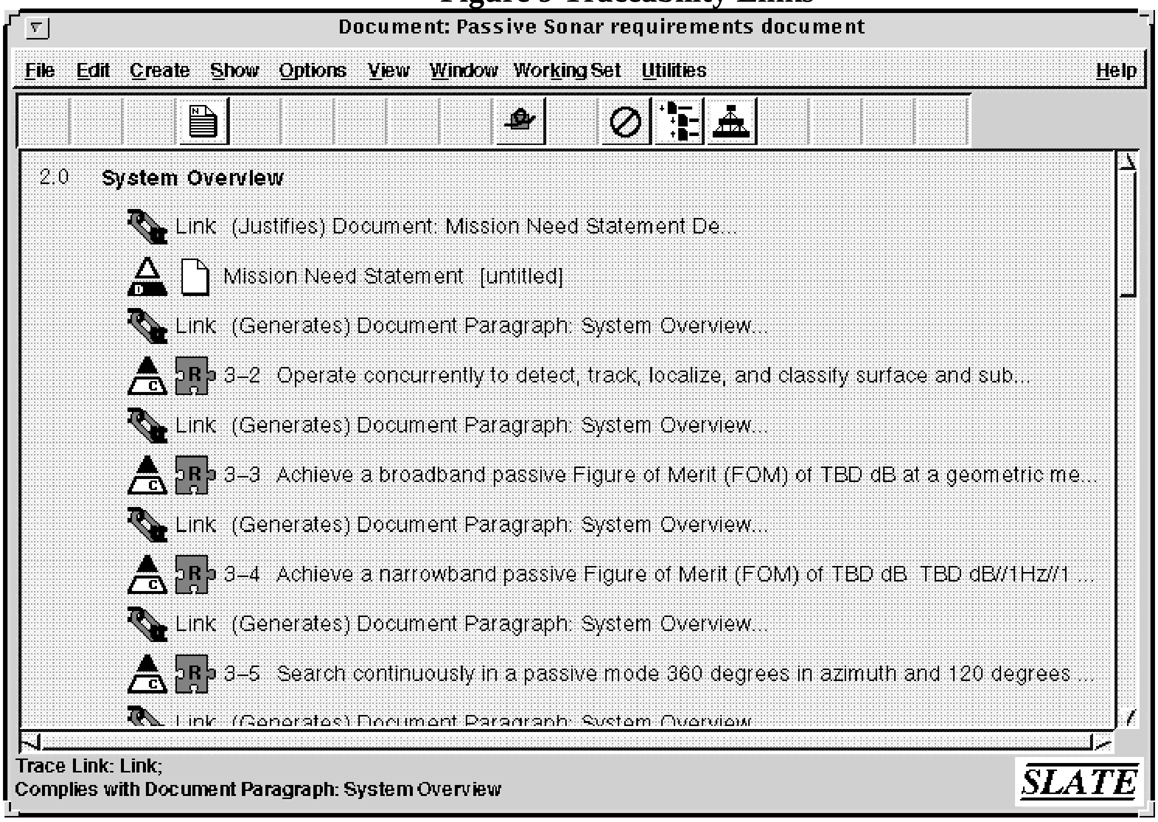


Figure 10: Requirements compliance with system objectives

Requirements Rationale. Figure 11 depicts information about a particular REQUIREMENT 3-3. It identifies the broadband figure of merit at a geometric mean frequency in a radio frequency band against a source level target. Requirement 3-3 has a link to the paragraph in the source document from which it was identified (see Line 5 in Figure 11). This source paragraph is identified as the defining object. In SLATE, RATIONALE can be attached to REQUIREMENTS in the form of notes. A user defined NOTE type called RATIONALE is used for this purpose. In our example, REQUIREMENT 3-3 is *SUPPORTED BY* a RATIONALE note (see line 7) This RATIONALE addresses the Figure of Merit (FOM) required to achieve sufficient detection range against most likely submarine threat. Requirement 3-3 is also linked to a paragraph from the system objectives that generated it.

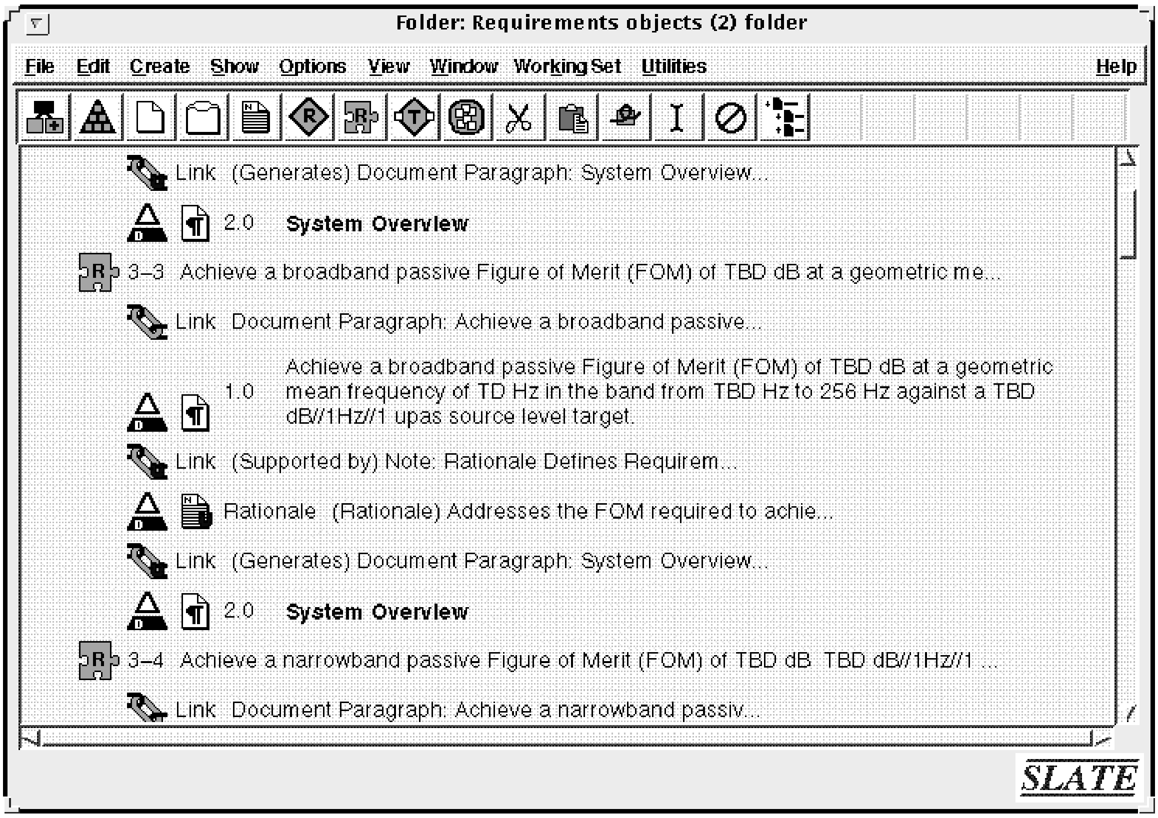


Figure 11: Requirements Rationale

6.2 Design Allocation

The design allocation model for high-end users has been tailored for use in the ECS problem. The model shown in Figure 12 identifies the linkages between requirements, design, system components, and functions. Further, it identifies resources used by system components as well as standards / policies/ methods that constrain the design. In this model, the link between standards/ policies / methods and design has been defined as the inverse of the link[4] shown between these objects in the high-end design allocation model shown in Figure 4.

[4] As mentioned earlier, for each link in our models an inverse may be defined.

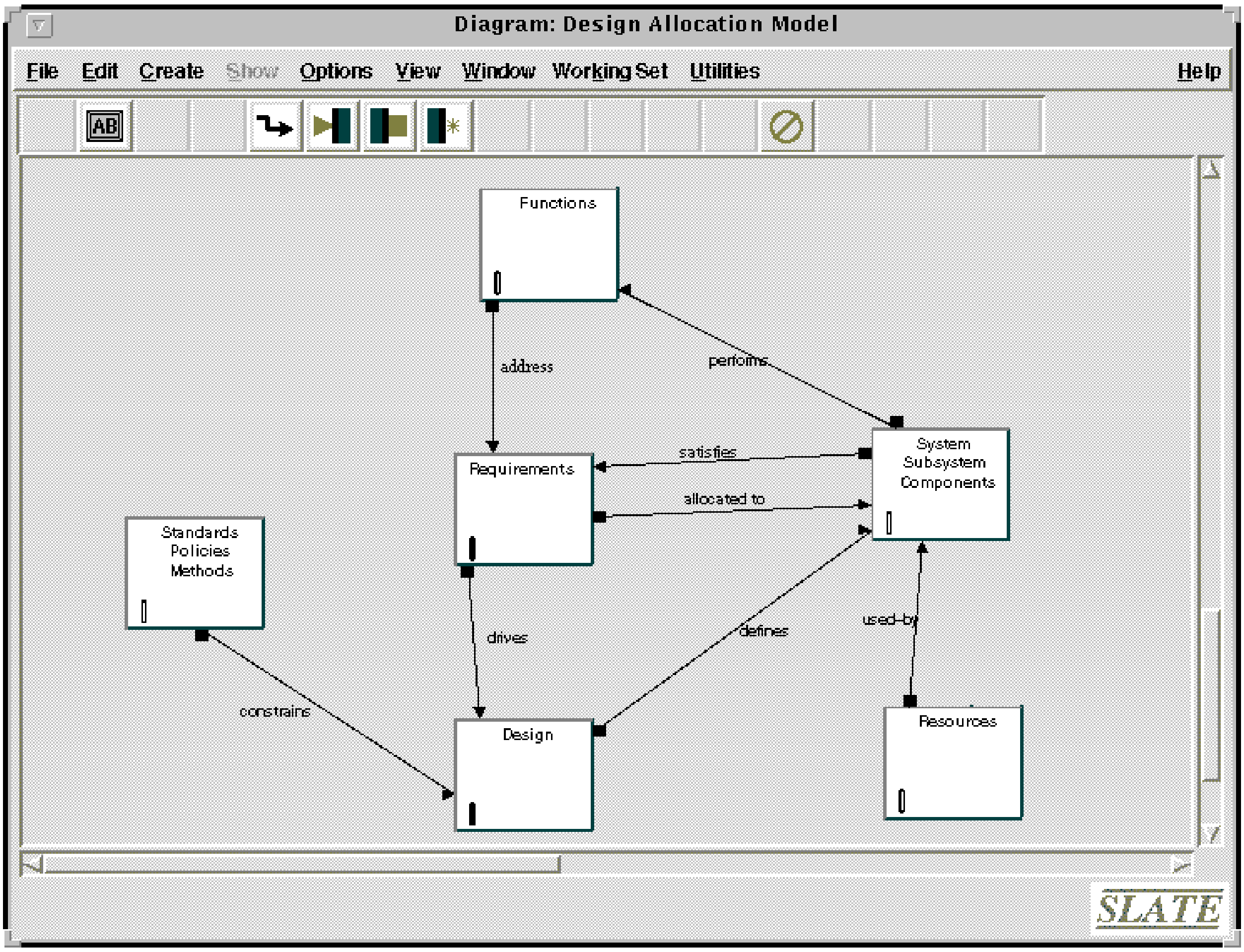


Figure 12: Design Allocation Model

During the design phase of ECS, Data Flow Diagrams (DFD) and Entity Relationship Diagrams (ERD) are used to capture the information flow in the system. SLATE provides an interface with TeamWork[5], a CASE tool used in ECS for this purpose. When abstraction blocks in SLATE are linked with the DFDs in TeamWork, objects called TeamWork handles are formed. Figure 13 illustrates abstraction blocks representing the sonar and its components as well as the TeamWork handles for the corresponding DFDs. Here the TeamWork handles represent the traceability link between information represented in SLATE and TeamWork. The SLATE objects are treated by TeamWork as requirements attached to individual processes in a DFD. SLATE provides a mechanism for synchronizing TeamWork when objects change in SLATE. A similar mechanism can be used to link requirements represented in SLATE and functions represented in TeamWork.

Figure 14 shows the defining objects for Requirement 3-3, namely its source document paragraph, its rationale, and system objectives. The first complying object (line 6) is a cell (Row 10, column 3) in a table that contains the constraints that must be satisfied by the design. These constraints were created by the design team based on the analysis of requirements. In fact, the value in this cell references a string in the requirements statement. Now, any changes made to the requirement will be automatically reflected in the constraint. In our example, the requirements statement mentions the need for BF microcode which in turn is referenced in the cell. If the requirement 3-3 is changed from BF microcode to SC microcode, it will automatically change the constraint on the functional design. Requirements in this project often

[5] Trademark of CADRE Technologies

contain such parameters that may get modified during development. Maintaining information in one location and referencing it in REQUIREMENTS (as well as other relevant objects), significantly increases the integrity of the information, besides reducing the possibilities of overlooking important changes.

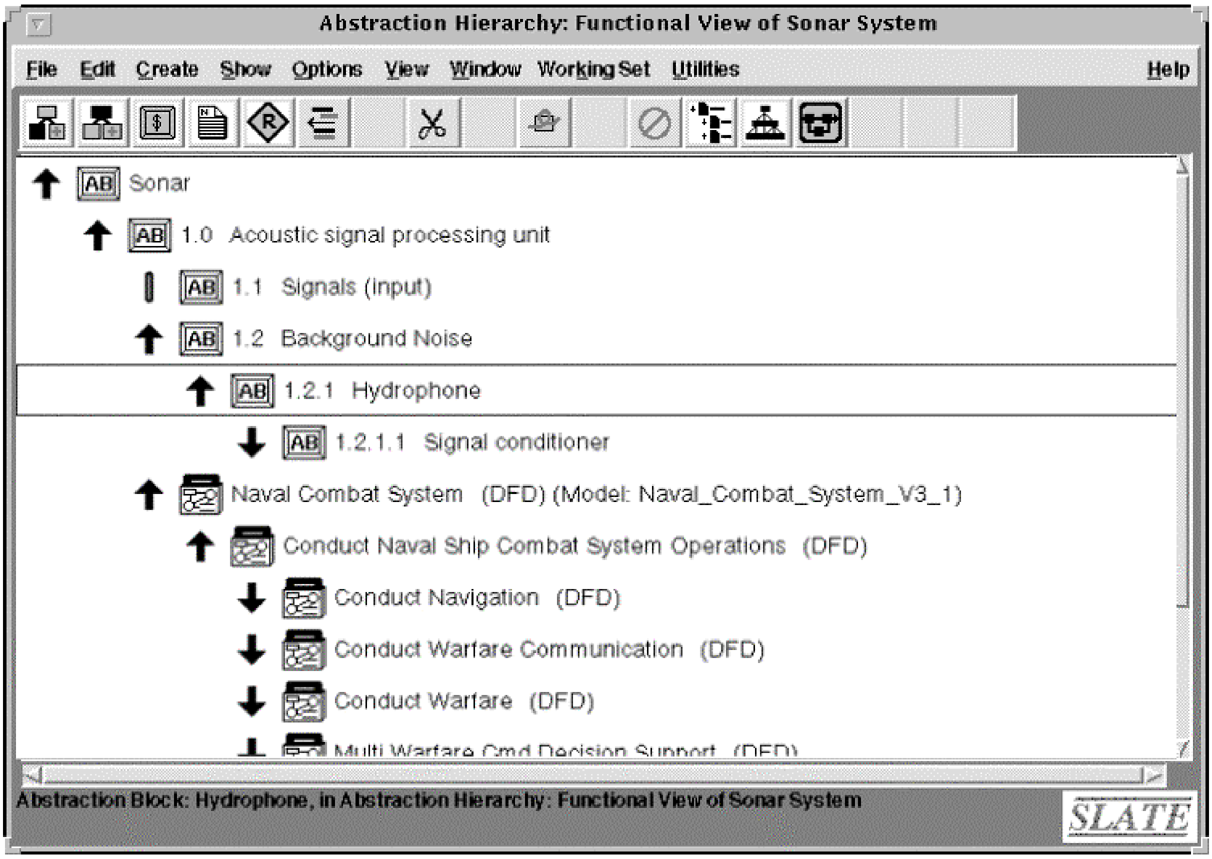


Figure 13: Design Linked to System Components

The next two objects shown in Figure 14 are two functions represented as abstraction blocks, 1.2.1.1 Signal Conditioner and 1.2.1.1.1 Digital Beamformer. The Requirement 3-3 has been allocated to these blocks that depict the functional view of the passive sonar system. These could just as easily describe any other view for instance, the physical system components to which the requirements are allocated..

In SLATE, CRITICAL SUCCESS FACTORS (CSF) such as resources can be modeled using budgets. In ECS, REQUIREMENTS are be managed by CSFs (as shown in Figure 7) and resources are used by system components. These are represented by attaching budgets to relevant objects such as REQUIREMENTS and SYSTEM COMPONENTS. Figure 15 shows that Requirement 3-9 is linked to a component (abstraction block 1.2.1 Hydrophone) to which a budget for hull size is attached. Budgets are used for planning as well as for performing "what if" analyses. SLATE supports hierarchical budgets (similar to nested spreadsheets). For example, the budget for the hull size attached to the Hydrophone can be linked hierarchically to the budgets for its children and so on. Any allocations done at the lowest level (leaf nodes) can be "rolled up" the hierarchy. Another potential use for budgets in complex projects such as ECS is the ability to monitor the cost and schedule of a project.

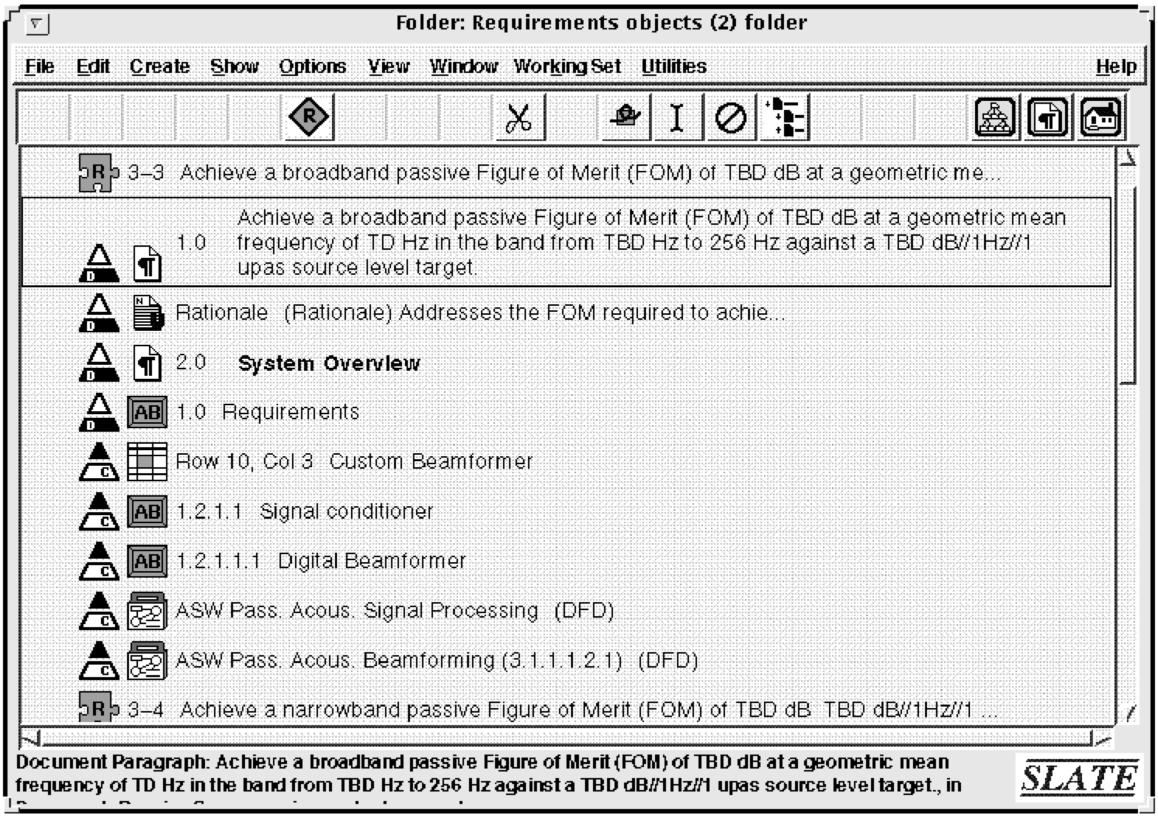


Figure 14:Traceability between Requirements and Design

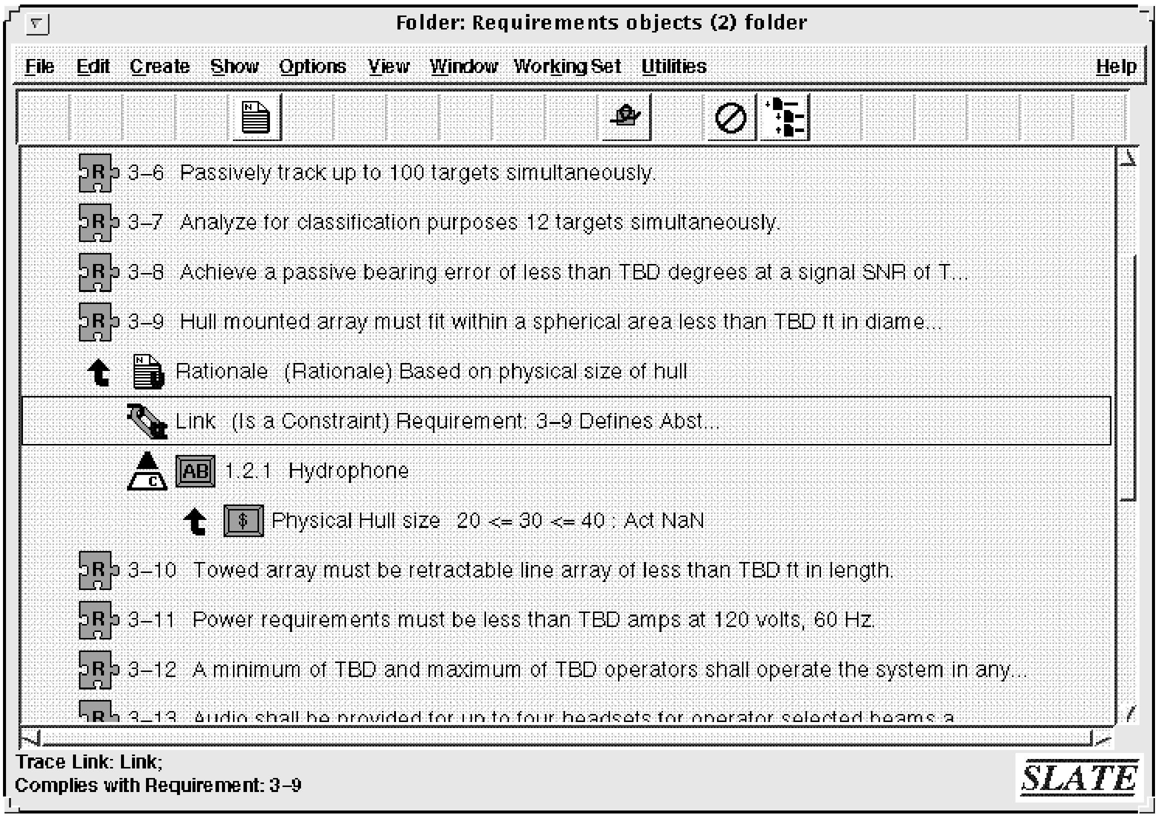


Figure 15: Budgets for modeling resources, critical success factors

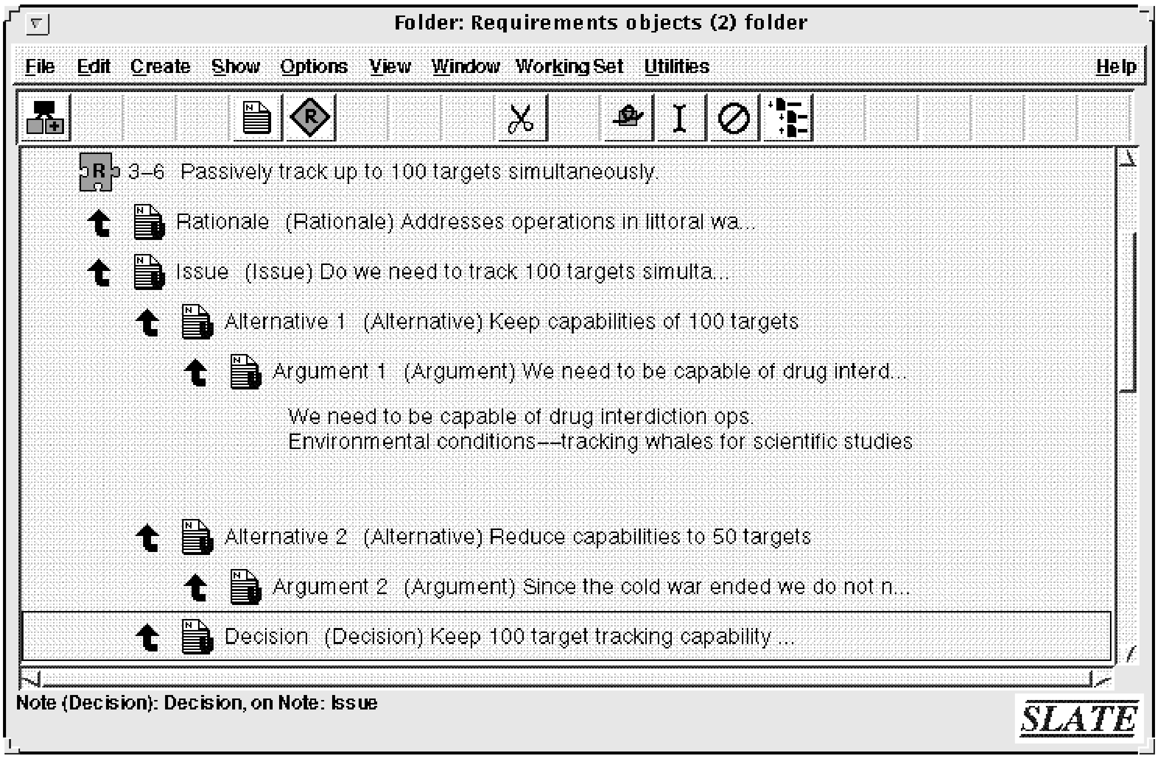


Figure 16: Rationale Capture

6.3 Rationale Management

Discussions about requirements may raise issues or conflicts. Alternative ways of addressing these issues are explored and decisions the stakeholders to resolve those issues are made. ISSUES, ALTERNATIVES, ARGUMENTS and RATIONALE (which are used to represent rationale - as shown in Figure 7) can be represented in SLATE using different types of notes.

Figure 16 shows an example. The ISSUE relates to the REQUIREMENT of tracking 100 targets. Two ALTERNATIVES 1 and 2 are proposed. Alternative 1 suggests that the system should "keep the capabilities of simultaneously tracking 100 targets." Alternative 2 advocates that the design "reduce the capabilities of simultaneous tracking to 50 targets." Argument 1 supports Alternative 1 by stating that "we need to be capable of drug interdiction operations …." Argument 2 supports Alternative 2 by stating "since the cold war ended, we do not need the capability of tracking 100 targets simultaneously." Based on these ARGUMENTS a DECISION is made to keep the capability of tracking 100 targets simultaneously.

Each NOTE type can be assigned attributes. In our case, attributes to identify the "originator" of the ISSUE and ARGUMENTS are used. Capturing this type of information is useful when a project is completed and the customer questions why something was implemented a particular way. Further, this information is useful is while performing system maintenance. Capturing the original issues and decisions can prevent rework on the same issues that have been resolved long ago. The mechanism described here can be used for maintaining rationale in any phase of the lifecycle.

In this section, we have illustrated how the high-end models proposed in Section 5 can be tailored and implemented in large-scale system design situations. This also provides a concrete example illustrating the instantiation of the model components. By using a commercial CASE tool, we further highlight the implementability of the reference models. In several such case studies, the models proposed in Section 5 have been shown to be both adaptable and implementable within currently available commercial environments..

7. Supporting the Traceability Links

Summarizing the differences between the low-end and the high-end uses of traceability, we observe that the latter are characterized mostly by two extensions:

- they employ a much richer type system for the product-related object and link categories, thus enabling easier retrieval and more precise (human or computerized) reasoning about traces
- they have a much stronger emphasis on the process-related categories; indeed, in the long-term case study conducted as part of this work (Ramesh et al. 1997), the step from low-end to high-end traceability was followed by a strong increase in the SEI capability maturity rating.

The large number of different link types precludes an individual treatment of each one. However, as shown in section 7.1, we can group the link types found in the reference models in a few basic categories. We then go again back to the empirical studies in order to identify the necessary tool support for each category, and point out proposals in the commercial or research literature that may address these issues. Conversely, this discussion then also provides a classification of the solution proposals advanced for traceability tool services.

7.1 Basic Classification of Traceability Links

Formally speaking, a traceability system can be defined as a semantic network in which nodes represent objects (also stakeholders and sources which we do not consider at the moment), among which traceability is established through links of different types and strengths. This dependency-directed approach of maintaining consistency of designs dates back at least to the work of Stallman and Sussman (1977) and is by now well established.

The simplest traceability tools are purely relational (i.e. in the form of relational databases or spreadsheets) and do not systematically distinguish different node and link types. Others, such as RDD-100, use a basic entity-relationship structure to distinguish nodes and links, and allow the user to introduce distinctions between different types of nodes and links. However, this begs the question which node and link types *should* be defined.

In the literature on software information systems, a large number of link types have been proposed, many of which are also important for traceability. The basic organization and abstraction mechanisms, independent of traceability, are detailed in (Constantopoulos et al. 1995). In the NATURE project, a comprehensive literature survey revealed eighteen different types of dependencies in the traceability context (Pohl 1996); these were grouped into five clusters named *condition, contents, documentation, evolutionary,* and the already-mentioned *abstractions*.

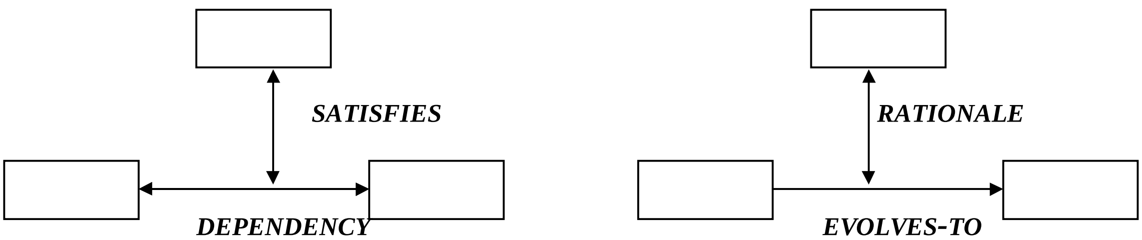


Figure 17: product and process related categories of traceability links

As a starting point for classifying our empirical observations, we adopt a simple system of just four types of traceability links, shown in figure 17. A first group of traceability links and mechanisms could be called *product-related* because they describe properties and relationships of design objects independent of how they were created. In figure 17.a, the high-level object (e.g. a requirement, a standard, a policy, or complex design object) defines some kind of constraint or goal which should be ***SATISFIED*** by one or more lower-level objects. Satisfaction is usually just a *claim* that must be substantiated by compliance verification procedures (cf. section 5). The shared constraint or goal to be satisfied also implies a ***DEPENDENCY*** between lower-level objects. Generalization and aggregation abstractions (i.e. configuration of complex objects) are special kinds of dependencies. Thus, we have two basic kinds of product-related traceability links, satisfaction links and dependency links; low-end traceability users tend to be characterized by relying mostly on these two link types.

The second group of traceability links is called *process-related* because it can be captured only by looking at the history of actions taken in the process itself, and cannot be recovered from the product relationships alone. Figure 17.b looks very similar to figure 17.a, with the important difference that the evolution link types has a *temporal direction* : the left lower-level design object ***EVOLVES-TO*** the right one through some kind of action whose ***RATIONALE*** is captured in the higher-level object. Thus, the two kinds of process-related links are evolution and rationale links. Advanced traceability users typically have a higher share of link types belonging to this category.

If we enhance the traceability meta model of Figure 1 with these four traceability link types, we get an 'onion-shell' meta model of traceability, as shown in Figure 18. Reflecting back to our empirical studies, the models used by low-end traceability users tend to focus on the inner kernel of the model, more advanced users tend to venture further out to process-related issue and more explicit source and stakeholder management.

As evidence for these observations, Table 5 summarizes the link types of the high-end models according to the way how and where they are used, and the link category they belong to. The abbreviations in the right column refer to the submodels in section 5 where the link types occur : RM for Requirements Management, R for Rationale, DA for Design/Allocation, and CV for Compliance Verification.

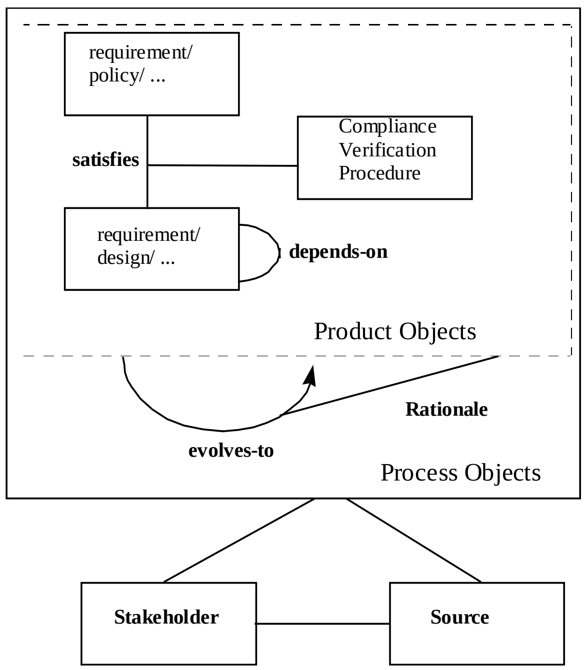


Figure 18: Traceability Meta Model Extended with Principal Link Types

LINK TYPE	PURPOSE	USES	EXAMPLES MODEL : LINK
Satisfaction Links	To ensure that the requirements are satisfied by the system. Relationships between one or more design, implementation objects and requirement objects verified by CVPs	to ensure consistency between outputs of different phases of the life cycle track the designs created to satisfy requirements track system/subsystem/ components to which requirements are allocated identify the system/subsystem/ components that satisfy requirements and that every component satisfies some requirement track CVPs that are developed for each requirement track the results of CVPs to ensure that the requirements are indeed satisfied	DA : Requirements-drive-design DA: requirements-allocated-system Component DA: Component-satisfies-requirement DA: system-performs-function & functions-address-requirement CV: CVP- developed for - requirement CV: CVP - verify - (system - satisfies - requirements)
Evolution Links	Document the input-output relationships of actions leading from existing object(s) to new or modified object(s)	Identify where do the various objects come from. I.e., identify the origins of various objects to facilitate: a) better understanding of requirements (and other objects) b) establishment of accountability for creation and modifications of objects c) tracking the modification, refinement history of various objects.	RM: system objectives-generate-requirements; RM: Organizational needs - justify-system objectives RM: scenarios-describe-{organizational needs |System objectives | Requirements} RM: Change proposals-modify-requirements RM: system objectives-generate-change proposals RM: requirements-drive-design RM: organizational needs-identify-CSF RM: requirements-basedon-standards CV: CVP-generate-change proposals CV: compliance verification procedures-based on-standards; R: decisions-affect-Requirements DA: design-based-on-standards/policies/ methods DA: change proposals-modify-design DA: design-defines/creates-component RM: requirements -derive-requirements RM: requirements-elaborate-requirements
Rationale Links	represent the rationale behind objects or document the reasons for evolutionary steps	identify the reasons behind the creation of various object to clearly identify a) justifications for the creation/ modification of objects, b) important decisions and assumptions made c) identify the context in	R: requirements-basedon-rationale; R: Requirements-generate-issues/ conflicts; Alternatives-address-issues/conflicts; arguments-support/oppose-alternatives; arguments-depend on-assumptions; R: {Requirements | Rationale | Decisions

LINK TYPE	PURPOSE	USES	EXAMPLES MODEL : LINK
		which objects are created d) provide transparency into the decision process including discarded alternatives, in order to - provide clear understanding of the current solution, facilitate maintenance and reuse - manage the system development with in line with organizational needs / objectives	| Arguments}- depend on-assumptions R: decisions-select/evaluate-alternatives decisions-resolve-issues/conflicts R: decision-influenced by-CSF RM: Requirements-managed by- CSF RM: system components - use - resources
Dependency Link	help manage dependencies among objects (typically at the same stage of development), often imposed by a constraint (resource, competence, compatibility)	track the composition and hierarchies of objects manage the repercussions of changes in one on other objects that depend on it	RM: {requirement | constraint }-ISA-requirement RM: requirement-part of - requirement RM: {strategic need | operational need}-ISA- organizational need RM: Resource-ISA-CSF CV: {Test | Inspection | Simulation | Prototype}-ISA-CVP RM: requirement-part of-requirement DA: component- part of - component DA: component-depends on-external system DA: component-depends on-component RM: requirement-depends on-requirement R: {Requirements | Rationale Decisions | Arguments}- depend on-assumptions

Table 5: Traceability Link Types Used in Reference Models

7.2 Issues in implementing traceability link types

Besides the classification of the different link types, a second goal of our study has been to identify issues in implementing their capture and use in system engineering practice. For each of the four link types discussed in the previous subsection, we first present the issues raised by our empirical study participants, and then discuss solution strategies proposed in the literature. The overall result is that local solutions exist for all the identified issues; however, the question how to configure such individual solutions to usable and more powerful trace tools than those today on the market remains open. In section 8, some more general design goals for such systems will be discussed, and an example of an interesting solution will be given where several of the issues addressed in this section have been combined in a creative manner.

7.2.1 SATISFIES Links

The focus groups raised three main issues concerning support for the satisfaction links:

Derived Links

An important use of requirements traceability is to ensure that the system meets the current set of user requirements. Representing this information is considered to be critical from a project management perspective. It is usually accomplished by creating a link between the set of requirements and a set of system components that ***SATISFY*** them. Such links may also be indirectly derived. An important concern of the study participants was the lack of support in many CASE tools for the automated identification of derived links ("I don't have the time to link every requirement with everything produced at different stages. These links must be automatically derived whenever possible"). For example, requirements may be linked to designs that are intended to satisfy them. Designs, in turn, may be linked to relevant system components. Then, a derived link is created from requirements to system components.

Mechanisms for automated inferencing incorporated in deductive or active database environments such as ConceptBase (Jarke et al, 1995) can be used to infer "derived links" in a traceability environment. As a critical first step, the semantics of the different linkages among objects must be clearly represented.

Multiple Sources of Support

When a set of requirements may be satisfied by a set of system components, the status of a requirement (whether satisfied or not) can be ascertained based on whether the relevant set of system components meet their goals. Most tools do not provide a way to represent the fact that alternate ways may exist for satisfying a requirement ("*we can't state that the requirement is met by either this solution or another one or a combination of system components easily in these tools. This will give me the flexibility to accurately convey the multiple solution paths I need to follow anyway*").

A goal tree is a good representation for answering questions about how goals were achieved and why they were attempted (Winston, 1984). Well defined inference procedures for finding "optimal" solutions from a goal tree can be used in finding the best possible way to satisfy a requirement or resolve an issue (say, with least cost) if they are represented as a goal tree. In the context of system engineering, the goals could correspond to requirements that need to be satisfied, issues that need to resolved or complex decisions that need to be made. Each of these "objects" in our model can be represented by a goal tree. The nodes of a goal tree are of two types: OR nodes represent choices i.e., they are satisfied when any of the immediate subgoals are satisfied; AND nodes represent simultaneous subgoals that must have compatible solutions; i.e., they are satisfied only when all subgoals are satisfied (Charniak and McDermott, 1985).

Degrees of satisfaction

During systems development, project managers would like to ascertain how close they are to satisfying critical requirements. In complex systems, many requirements (especially, non-functional requirements) can not be said to have been "satisfied" absolutely. Often, a designer is interested in maximizing the degree to which a requirement is satisfied. For instance, a performance requirement to provide response time of 100 milliseconds in a missile system may be thought to be reasonably well satisfied if the system meets the requirement within a few milliseconds of this target. Here, 105 millisecond and 95 millisecond response times may be considered to satisfy the requirement with different degrees. Most current tools do not provide natural ways to represent such information. ("*For compliance, I want to track how well I am doing on a requirement based on progress in implementation. It is hard to get this information into and out of my tools'. I need this information for trade-offs*").

Mylopolous et al (1992) suggest that goal satisficing (rather than satisfying) may be a more appropriate way of characterizing this situation. The concept of satisficing, as defined by Simon (1981), refers to methods that look for satisfactory rather than optimal solutions. Using this framework, a system component may be thought of as contributing positively or negatively towards satisficing a requirement. Hauser and Clausing (1992), in the House of Quality, use four categories to relate how each development characteristic affects the customer quality requirement: strong positive, medium positive, medium negative, and strong negative. Here, we have a measure of not only the directionality (positive or negative) of a relationship, but also its strength (high, medium etc.). This scheme can be incorporated in our traceability model as follows: a system component may contribute towards satisficing a requirement along these four (or a finer grained) categories. For example, a user interface component may have a strong negative impact on a performance requirement..

All three issues – derived links, multiple support, and degrees of satisfaction – can also occur in combined form. When a requirement is satisfied by a set of component, mechanisms for combining the "evidence" for its satisfaction must be developed. For example, if all the "evidence" is in the same direction (positive), the degree to which the requirement is satisfied may be treated as the minimum of the support it receives. Situations in which a requirement receives both positive and negative support at varying degrees pose a significant problem. Requirements that rely on user perceptions (such as usability of a system) require qualitative measures of ascertaining compliance as well as qualitative mechanisms for combining evidence. Combining beliefs in evidence and propagating such beliefs in such an inference network may be supported with mechanisms such as Dempster-Shafer, Bayesian Probability, Belief Networks and Fuzzy Logic (Hau and Kashyap, 1990), (Hussain et al, 1994). Whereas Fuzzy logic provides good mechanisms for representing and combining imprecise, qualitative evidence, the other, more quantitative methods provide stronger methods for combining evidence from multiple sources.

7.2.2 DEPENDENCY Links

The dependencies created between objects when trying to satisfy some goals are at least minimally supported by almost all traceability models. Issues raised by the study participants therefore address the lack of precision in characterizing the types and strength of such dependencies, resulting in much less automated support than would be possible.

Types of dependency and inferencing services

Most traceability models explicitly represent dependencies among objects. For example, dependencies among system components, requirements, assumptions and decisions are part of a high-end traceability scheme. The type of dependencies that exist among objects vary widely. For example, a software requirement may depend on a requirement for a specific hardware, implying that in order to meet one the other has to be met. Such a dependency is very different from the relationship between two outputs that must be produced in a pre-determined order. For example, an organization may have a policy that the test plans for a requirement must be developed before design solutions that satisfy the requirement are considered. Our empirical studies identified two problems related to the management of dependency links. First, most traceability frameworks simply identify that there exists a dependency between a set of related objects without the ability to specify the nature of the dependency *("We create matrices that show that there is the hardware requirements on one column that affect the software requirements on the other. This obviously is not good enough. It is not clear whether these two are related as they consume the same resource like power or because one has to be completed before the other.. One needs to intuitively figure out what the connection is.*"). Second, as the nature of the dependency is not well specified, the ability to provide support the use of traceability information is limited. ("*We got this dependency as the two components will have to share the same memory. As I keep developing one, the impact on the other must be easy to figure out .. how much the total memory is already accounted for. But my trace matrix stops with just listing the two involved components. There is no help for assessing how changing one affects another*"). Thus, our study suggests that the ability to define different types of dependencies within a traceability framework and developing support mechanisms to manage such dependencies will be very valuable.

In addition to research in theoretical computer science and in operations research. several schemes for identifying different types of trace-oriented dependencies have been proposed in the literature. For example, Yu and Mylopoulos (1994) classify dependencies among stakeholders into three categories: goal dependency, task dependency and resource dependency. This scheme can be adapted and extended to manage dependencies as follows:

- Dependency links among requirements may often be *goal dependencies*. For instance, a software requirement may depend on a requirement for a specific hardware. When a goal dependency between two such requirements exists, inference procedures that monitor whether (or how well) the dependent requirement is satisfied are necessary:
- The dependency between objects created by their common reliance on a task *creates a task dependency*. Design objects (say, the ERD and DFD) often have *task dependencies*. Here, the corresponding artifacts must be produced following a specified method (e.g., structured methodology). If the object (say, ERD) is produced using any other procedure, the correspondence between the two components may not be as expected. When a task dependency exists between two objects, procedures for monitoring the progress of the task using the specified procedure are essential. For instance, whenever a change in one of the artifacts is initiated, all associated task dependencies must be checked for compatibility.
- A *resource dependency* may involve resources that are physical (e.g., money, electricity) or informational (e.g., data). A navigation control system may resource-depend on radar input. When such a dependency exists, the quality, quantity, and frequency of such dependence and the interfaces for the transfer of resources must be identified. When a resource dependency exists, procedures for monitoring that the dependee provides the desired quality and quantity of resources at specified frequency are necessary. It may be necessary to identify who is responsible for the resource and the conditions on which the resource delivery depends.
- Additionally, a temporal *dependency* exists when actions relating to objects are done in a specified temporal order. The temporal order could be specified by operators such as before, after, during, concurrently etc. Often system development activities involve strict temporal ordering. For instance, a temporal dependency link could be set up to enforce the policy that test plans must be created before designs for addressing a requirement are completed. Violations of temporal dependency constraints can be monitored using a temporal reasoning system.

Strength of dependency

Another problem associated with maintaining dependency information is that all dependencies are treated as equally critical. Our studies identified the inability to precisely identify the strength of a dependency link to be an important issue for the users of this traceability information. ("*we got two requirements in this matrix, all I know is that changing one will affect the other. In some cases, I really need to take it serious. The impact can be devastating. But often times it is not that serious. But I got to ask around to see which ones I should really worry about. It's not in the matrix*").

Hauser and Clausing (1988) propose a scheme for assigning qualitative or quantitative weights to links. Yu and Mylopolous (1994) adapt this scheme in their non-functional requirements framework. Based on these, the degree to which a dependency is important to those involved in a dependency linkage is termed the *strength of the dependency*. This is a measure of how much one object affects the other. It could be measured qualitatively (e.g., high, medium, low etc.) or quantitatively (e.g., 1-10 on a 10 point scale). The strength of the dependency will also very much influence the type of inference procedures and monitoring necessary to ensure the enforcement of various types of dependencies.

A network of dependencies such as those discussed in this section can be defined at different levels of formality (Stallman and Sussman 1977). At the most basic level, a link simply identifies the existence of a dependency between two objects. As the only information contained in such a representation is that one object has something to do with the other, the potential for automated reasoning is limited, even when the type of dependency is identified. For example, if there exists a resource dependency between say, two components in a satellite, we can infer that whenever the weight of one changes, it will affect the other component. A more detailed representation will also capture the direction in which the dependency affects an object. In this example, information on whether the effect of the change is positive or negative can be captured. Finally, a very detailed model will represent the exact nature of the influence that one component has on the other, possibly expressed in a detailed, domain-specific mathematical model. It may, then, be possible to compute the effect of change in the weight of one component on the other. Obviously, the more well defined the dependencies are the more sophisticated the inferencing capabilities.

7.2.3 EVOLVES-TO Links

Traceability models represent the process of modification, refinement and evolution of different objects through a variety of links. The objective is to capture the history of the development in a structured manner so that it will facilitate understanding and management of the objects. For example, in our high-end models, requirements evolution is modeled using links such *as MODIFY*, *DEFINE*, *ELABORATE*, *DERIVE* etc. These links can be considered to be specializations of some generic *EVOLVES-TO* link. Our empirical studies highlighted two dilemmas in the management of links representing evolution. First, many traceability frameworks simply represent the fact that an object has evolved, without any elaboration. For example, derived requirements may be simply identified as a new requirement *("don't know where the derived requirements came from*"). On the other hand, very fine-grained representation of the evolutionary path may not be desired in cases where the process is implicitly well understood. ("*We manage derived requirements in a separate document. As long as we create a simple trace matrix showing the customer requirements and the derived requirements we are fine*.") Another area of concern is the weak support for versioning and configuration management in traceability tools *(" I have good sense of how my source code is managed within a configuration system with nice features for notification of changes, but not with my traceability tool*").

When a fine grained representation of the evolutionary path of an object is not desired, a higher level view can be captured by using just the generic link. Much of the pre-requirements traceability information can be thought of as representing the evolution of needs and goals into requirements and therefore, can be represented using these links. The evolutionary links can be used for chronological replay of the evolution of an artifact. For subsequent phases of development, transformational approaches to software engineering, constraint-based techniques as well as research in version and configuration management have created elaborate refinements (and associated reasoning schemes) of the simple *EVOLUTION* link. We do not pursue these specializations further in this paper, but just mention that they can be easily embedded in the framework presented here.

Software configuration management (Conradi and Westfechtel 1998) tools provide facilities for representing and enacting policies regarding the evolution of coarse-grained artifacts, and provide mechanisms for notification of the user about changes. These functionalities are likely to be very useful for managing traceability information. However, traceability tools often manage finer-grained objects with more representational diversity, so the combination of configuration management and traceability remains a partially open issue.

7.2.4 RATIONALE Links

Issues concerning the rationale links partially repeat the demand for multi-granular representations already discussed for other link types; however, in addition, the capture of the stakeholder dimension of traceability is considered a key issue.

Rationale at different levels of detail

Study participants wish to maintain rationale for different types of artifacts and decisions at different levels of detail. The criticality or the importance of a requirement, for instance, will be a principal factor in deciding the granularity what rationale needs to be captured. In contrast, many traceability schemes we observed in practice prescribe a single framework for representing rationale which is either too fine-grained to justify the overhead in all cases or too coarse to be of value in critical instances. ("*We are supposed to be documenting critical decisions. We are just now going through a review of a very, very critical decision.. worth several hundred million dollars. We can not for the life of the project figure out how we came to the conclusion we came to in the first place. It looks doubly bad with the sponsors that we come up with a different answer now. It's all in the details and assumptions we made. But this is where good traceability could have paid off big time .. But we can't do it this much detail with every decision. We just can not afford the time for so much detailed documentation for every case… How do we do it right - It is a major issue*") ("*We got all these details. As a manager, I can't get a quick glimpse of the big picture from there reports easily*").

Most tools for the management of rationale (such as gIBIS, DRL, SYBYL) as well as traceability tools such as RDD-100 support a uniform approach to the representation of rationale. This results in either a very simple scheme that is not detailed enough to document critical aspects of systems development or a comprehensive scheme too complex to be implemented in every stage of systems development. Also, a detailed study (Ramesh et al, 1997) reveals that when the amount of rationale captured is left to the discretion of the project participants, the quality and content of this information fluctuates depending e.g. on time pressure and personal preferences. The rationale management approach taken in the KBSA-ADM tool (discussed in Section 8.4) illustrates a compromise. This tool supports the capture of rationale at multiple levels of detail and also provide a way to map the rationale captured with a comprehensive rationale into a simple view to provide a "*big picture*".

Link to sources and stakeholders

Rationale links help represent the context in which various objects are developed. Our empirical study highlighted two common concerns: The need to link to the various sources of this information to not only reduce the overhead in capturing rationale, but also to ensure minimal loss of detail ("*As long as I know where to look if I want to understand why we decided the way we did, it is sufficient. I not only am too busy to reenter into the trace tool, also I don't want to interrupt the critical activities to do it*").

In large projects, the overhead involved in the detailed representation of rationale can be considerable. Facilities to link any object (say, a requirement or a decision) to its sources such as documents, meeting minutes, e-mail exchanges etc. will help minimize the overhead. Also, this facilitates non-intrusive capture. The use of multiple media to capture such informal knowledge has been long recognized in the literature (Goldman and Segall, 1992, Mead, 1975, Rochelle et al, 1990), and confirmed in studies of design situations e.g., (Baudin et al, 1990, Lakin et al, 1989, Sults 1988). The use of thick descriptions for representing traceability is discussed in detail in Section 8.3.

8. Implications for Traceability Mechanisms

Traceability mechanisms support the capture and usage of trace data, i.e., to document, parse, organize, edit, interlink, change and manage requirements and the traceability links between them. Besides generic tools such as word processors, spreadsheets, hypertext editors, and relational databases, a number of specialized RT tools are being offered, either stand-alone (e.g. RDD-100 (Alford 1991), DOORS (QSS 1996), SLATE (TD Technologies 1996)) or embedded in an integrated CASE environment (e.g. Teamwork/TqT (CADRE 1992)).

The majority of present traceability tools offer tabular representations of dependency structures in a relational database formalism. Thus, they are well suited to support the low-end user who requires little differentiation between link types and only very simple allocation and dependency analysis. A few tools allow the user to specialize link types but it is hard to attach semantics to them. Others offer a fixed high-end set of link types with hardcoded semantics but tend to force the user to actually supply this detailed information in all cases, even if a simpler model would suffice. Moreover, most tools just offer mechanisms for persistent storage and display of traceability information, but they do not support the *process* of capturing and reusing traces by guidance or enforcement in a systematic manner; those that do tend to have very rigid process models.

Not surprisingly, our empirical studies showed that users are not very happy with the current mechanisms supporting traceability. Some high-end development teams have built elaborate in-house extensions and work-arounds to existing tools which, however, also tend to become inflexible and difficult to maintain. Even high-end users do not always use the sophisticated models presented in section 5. Instead, as indicated when discussing the various link types, they carefully consider the trade-off between the effort needed to capture complex traceability information and the subsequent value of having this information in each situation in the development process. These observations indicate that a new generation of traceability mechanisms will be needed which we shall try to characterize in this section.

The key observation is the *situated nature of traceability capture and use*. The implications of this observation include:

- the need for *abstraction mechanisms* that allow the variation of granularity (aggregation abstraction) and sophistication (generalization abstraction) in traceability tasks
- the need for *inference* services supporting the semantics of the different traceability link types
- the need for maintaining a baseline of *thick descriptions* of traces (e.g., using multimedia), in case the initially chosen level of trace analysis needs to be re-assessed later on
- the need for mechanisms that allow project managers and developers to define and enact a *model-driven trace process* accompanying the development process.

We discuss these implications in turn. In the final subsection, the design of the KBSA ADM tool by Andersen Consulting illustrates how some of these principles have been applied in practice.

8.1 Abstraction Mechanisms in Trace Representation

The relational representation used in most existing tools does not lend itself naturally to the simultaneous representation of traces at multiple levels of granularity (e.g., milestone level, document level, fine-grained level). The act of representation entails making judgments about the level of granularity at which the information should be represented. Overly large grained representations may result in the loss of useful detail, while overly fine-grained representations may create trivial knowledge where the benefits do not warrant the cost of creating such knowledge. Aggregation hierarchies in semantic data models offer an adequate formal means to deal with the granularity problem. However, existing literature does not offer demonstrably effective decision rules for making judgments about granularity; therefore, tools should support multi-granular strategies such as presented in section 8.4.

A similar problem of over-commitment is faced when considering the generalization abstraction. Such hierarchies allow a smooth transition from low-end to high-end traceability models, at least in the sense that a mix of traces captured with varying sophistication can be interpreted uniformly in terms of the low-end models used.

Finally, the instantiation abstraction allows the definition and change of meta models, and thus the adaptability of traceability environments to new needs via re-organization and possibly the integrated interpretation of traces captured in heterogeneous environments.

In summary, we advocate a move towards more semantics in traceability data models. This move will be facilitated by recent trends in object-relational database technology that offer many of the required extensibility features on top of the proven framework of the relational data model.

8.2 Inference Services

In large projects, the traceability knowledge base can become quite large. As the information needs of various participants can vary widely, navigating the entire knowledge base to retrieve relevant information can be very difficult even with sophisticated browsing capabilities. Ability to access relevant information using ad-hoc queries can be very helpful.

Inferencing also helps maintain integrity of a knowledge base of interdependent components that get incrementally defined and modified. Such inferencing will be aided by mechanisms for aggregation, classification and generalization of traceability information.

The ability to make deductive inferences from traceability knowledge base is very helpful in reducing the overhead and increasing the usefulness of the traceability knowledge base. Further, mechanisms to maintain and manage dependencies among various components of requirements traceability can be supported with deductive database capabilities for query processing, deductive rules and integrity enforcement.

8.3 Grounding Traceability Models in Thick Descriptions

Requirements traceability can be represented in a variety of ways, from *mathematically* formal representations (e.g., transformations that can be used to derive one problem state from another in formal software development) to very informal representations (e.g., design notebooks that record rationale in natural language) (Lee 1993).

A primary benefit of formal representations is that they facilitate automated reasoning. However, formal representation of requirements traceability is often difficult as most design situations lack well-defined formal models. Converting informal traceability information into formal models is often extremely labor-intensive and subject to personal bias and choices.

Since design is primarily a collaborative process (Fischer et al. 1992), requirements traceability knowledge often consists of deliberations among individuals engaged in the process. As Anderson et al (1991) point out, attempts to represent informal knowledge through formal tools and notations can result in thin descriptions (Ryle 1949), with the consequence that much of the meaning embedded in such information is lost.

Informal representations of requirements traceability can address many of these problems. Such “thick descriptions” (Geertz, 1973) enable the user of requirements traceability to grasp the subtleties, tacit and mutual knowledge, and glean descriptions of work practices that are otherwise not made explicit (Jordon, 1992). Finally, unlike formal representations, which can only be used by individuals who are familiar with the rigors of such formalisms, informal representations can be used by a wide set of users. However, informal representations provide another set of problems: the classification, indexing, retrieval and use of such information in large projects can be impractical. Moreover, though understandable by humans, such information representations are not amenable to automated reasoning. Thus, while informal representations hold much promise with respect to their information content and ease of capture, the difficulty of accessing and reasoning with such information can undermine their utility.

In summary, then, formal and informal representations of requirements traceability complement each other in their respective strengths and weaknesses. Informal representations are easy to capture, whereas formal representations can be manipulated by well-defined inference procedures. Thus, as observed in recent studies on design rationale (Shum and Hammond 1993), effective schemes for the capture and use of requirements traceability should seek to combine the advantages of both forms of representations. A tight integration of formal and informal aspects of the requirements traceability information will not only facilitate the capture of various types of information in a format that is most appropriate, but also help use/reuse the captured information in an efficient manner.

8.4 Model-Driven Trace Capture and Usage

Assuming the above-mentioned formal and representational problems have been solved, the project manager or developer can now define the kind of information to be recorded in a trace, i.e., specify the products of traceability. However, the traceability environment still lacks two other important features:

- It provides no means to define the *trace steps* by which the defined information should be recorded. For example, the project manager can define that decisions should be recorded together with their arguments. But she cannot define in which situation and by which specific process steps this should happen. In other words, we cannot define the process guidelines according to which the trace information is captured or used during the development process.
- It provides no tool support for the stakeholders in actually capturing the information according to a defined trace process. Depending on the importance attached to certain trace information in a given situation, the environment could either simply wait for the information to be entered optionally (the default option in most current tools), it could remind the stakeholder to record the information, it could force the stakeholder to do so by blocking the further development process, or it could even record (some of) the information automatically. The development of such an approach requires that the traceability process cannot just be defined but also effectively embedded in the development process and its supporting CASE environment.

Dömges and Pohl (1998) describe a prototypical environment in which selective, model-driven trace capture is possible. Applied to the modeling approach taken in this paper, the approach works roughly as follows. Each basic element in the traceability meta model is seen as a product, and associated with process steps that create, delete, or change this product.

In addition to these normal process steps, there are three additional kinds of trace steps. *Supplementary product steps* define how to capture and maintain supplementary information such as design rationale, *process execution steps* record activities in the development environment, and *dependency steps* record dependencies between the artifacts created by process steps, supplementary product steps, and process execution steps. While process execution steps and dependency steps can often be performed largely automatically, by simply observing the process in the development environment, the other two kinds of steps usually require human intervention. Based on these conventions, the project manager is thus empowered to define

- the trace information which should be captured
- the three kinds of trace steps defining *how* this information should be recorded
- the interleaving of the trace steps with the normal process steps, i.e., *when* the trace information is to be captured.

In a sufficiently flexible process-centered software engineering environment, the thus extended process can now be enacted, thus guaranteeing the desired traceability sub-process. However, it must be noted that there are very few process-centered software environments on the market at the moment, so the practical feasibility of this approach can only be assessed from a few case studies conducted with the available prototypes.

8.5 An Example: Multi-Granular Rationale Capture in KBSA ADM

In this subsection, we use the Rationale Capture part of Andersen Consulting's KBSA ADM tool to illustrate the practical application of the principles discussed in this section : the role of multi-granular abstraction, of inference services, formal and informal representations of traceability, and to some degree of model-driven trace capture and usage. The combination of these principles is mainly aimed at achieving systematic customizing of traceability needs spanning the distance between the high-end and low-end reference models.

Abstraction in Rationale Models. The design goal of KBSA-ADM was to offer a coherent series of rationale models based on results of the REMAP project (Ramesh and Dhar 1992) for maintaining rationale at different levels of detail.

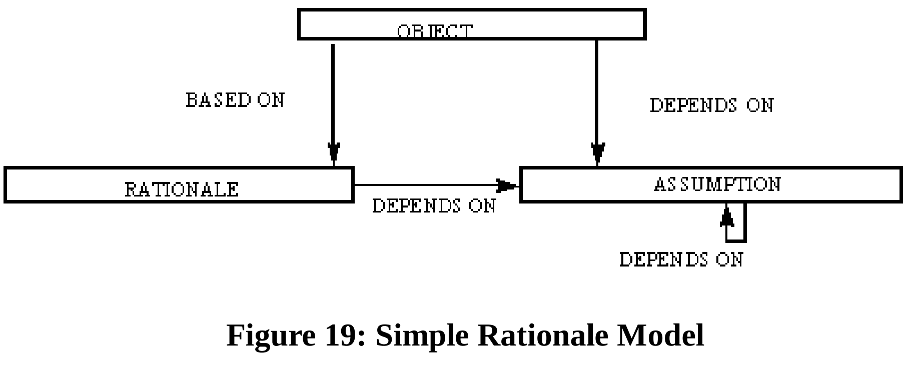


Figure 19: Simple Rationale Model

The model sketched in Figure 19 is used for capturing rationale at a simple level of detail. It links an OBJECT with its RATIONALE. The model in Figure 19 also provides for the explicit representation of ASSUMPTIONS and *DEPENDENCIES* among them. Thus, using this model, the assumptions providing justifications to the creation of objects can be explicitly identified and reasoned with. As changes in such assumptions are a primary factor in the evolution of complex systems, this representation will help in the development of support mechanisms to manage such evolution.

In situations where a more detailed representation of rationale is necessary, an intermediate rationale model may be used. In this model, the ISSUES that are *GENERATED* by the OBJECT as well as DECISIONS that *RESOLVE* them are explicitly represented. Thus, this intermediate model provides some elementary details of the decision process involved in creating various objects. As before, the ISSUE and DECISION objects can be complex with attributes of their own. Therefore, using this model, a richer context in which objects are created can be captured.

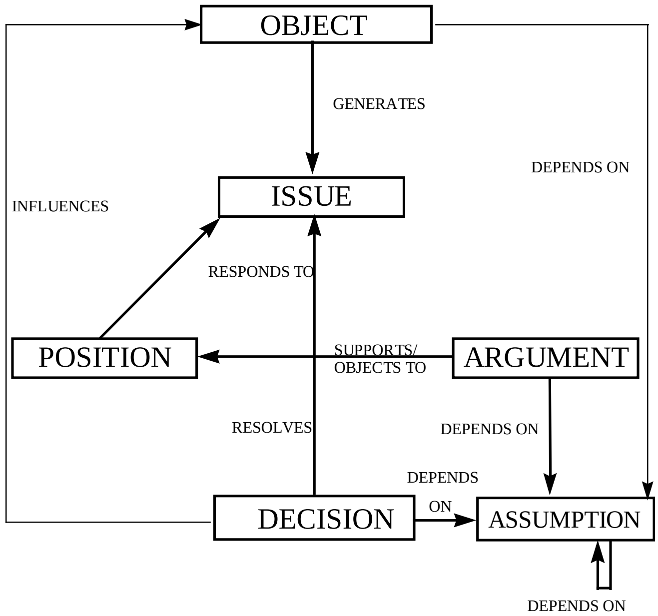


Figure 20: Detailed Rationale Model

An even finer grained model for representing the detailed deliberations leading the creation of objects may be more appropriate in critical situations. Empirical studies in the REMAP project (Ramesh and Dhar 1992) suggest that the explicit identification of ALTERNATIVES considered, the ARGUMENTS for and against these ALTERNATIVES, and ASSUMPTIONS behind such ARGUMENTS themselves are part of such a deliberation. A detailed rationale model incorporating all of these primitives is shown in Figure 20. It incorporates, among other concepts, the Issue Based Information Systems (IBIS) framework (Conklin and Begeman, 1988). Such models have been used successfully in large scale system development efforts to represent complex decision situations leading to the creation of artifacts.

The components of this detailed model also closely resemble the Rationale submodel presented in Figure 4. All the objects used in this model are represented in the Rationale submodel and the major links across these objects are also identified in that model.

The hierarchy of simple, intermediate, and detailed rationale models have been re-implemented in Anderson Consulting's Knowledge Based Software Assistant environment ADM. This implementation has the added advantage that different stakeholders may be interested in different depth of rationale; for example, designers might use the detailed rationale models whereas their managers will use the simple rationale view of the same model focusing just on the decisions finally made. Two KBSA-ADM screenshots, using an example from (Ramesh and Dhar 1992), are shown in Figures 21 and 22.

Inference Services. As the model components have been defined with formal semantics, a system based on these components can support sophisticated reasoning with rationale information. For example, a dependency network of assumptions is managed using a truth maintenance system which propagates the effects of changes in the beliefs in assumptions to other components of design rationale knowledge. In the detailed model, such a mechanism will help identify the decisions that need to be reevaluated when changes in assumptions underlying the network of rationale change. KBSA-ADM also supports truth maintenance on such structures, using not just the simple logic-based approach, but also the Dempster-Shafer theory of evidence to determine the belief status of the decisions made.

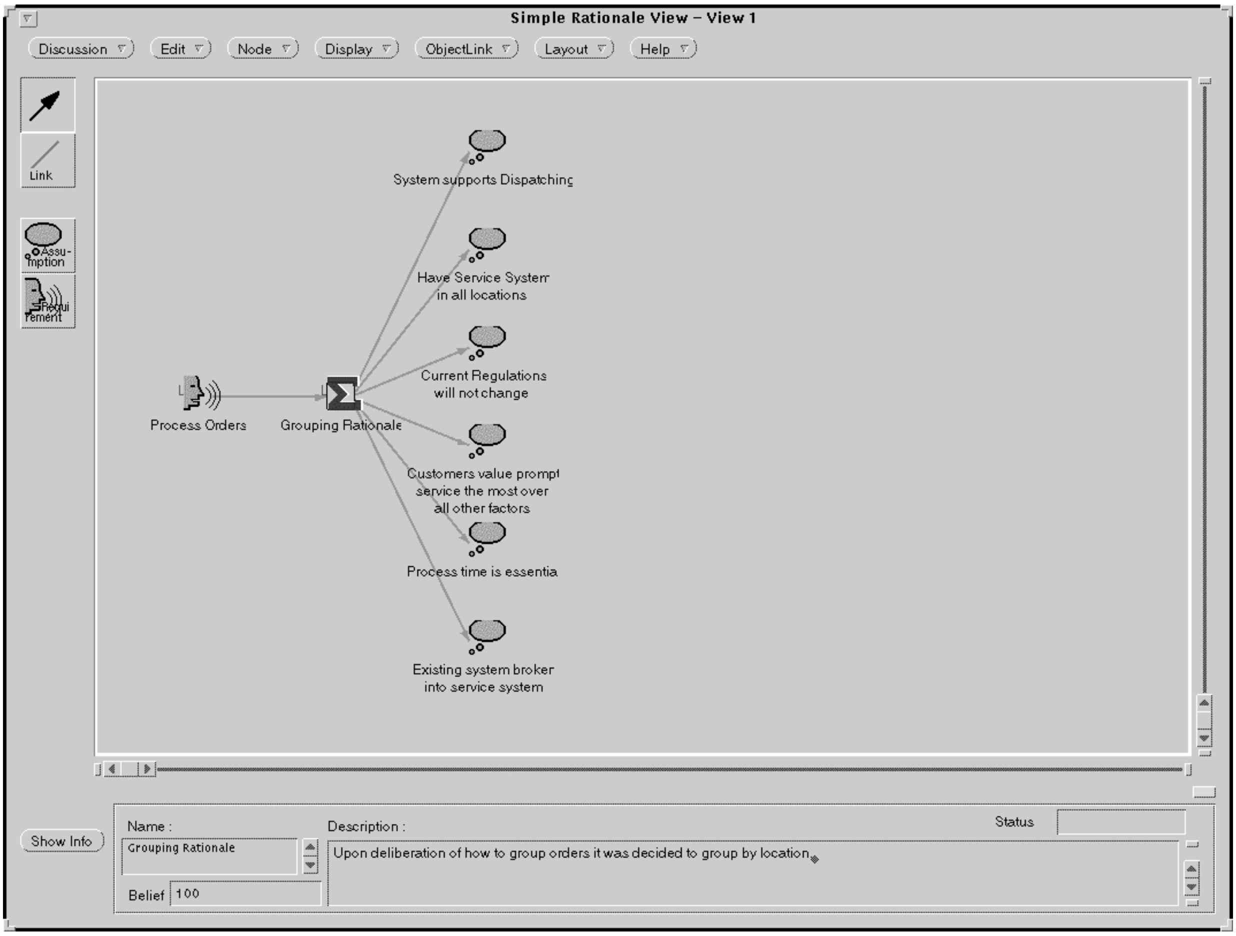


Figure 21: Rationale in Simple View

Thick Descriptions. KBSA-ADM provides a few facilities for integrating formal and informal components of traceability knowledge. First, each object in a discussion can be annotated with multimedia clips. For example, in the discussion involving the processing of orders in a utility company shown in Figure 22, the node representing location (i.e., graphical location of various services provided by the utility) is annotated with a map of the service area. Further, the tool provides a hypermedia editor within which various objects in our model can be created by simply highlighting a section of a hypermedia document. Such an integration also facilitates easy capture of traceability information.

Model driven capture. KBSA-ADM provides mechanisms for specifying, executing and monitoring process steps. This can be used to define the steps involved a system engineering process and the conditions under which such steps should be executed. Such a facility can be tailored to define the various types of traceability information that should be captured and used. For example, the facility can prompt the user for design rationale information after major

design decisions. An agenda mechanism is used to monitor compliance. However, the current implementation does not include comprehensive model-driven traceability capture.
In summary, the KBSA-ADM example illustrates that the advanced tool features discussed in this section are indeed feasible in industrial settings. However, the example also shows that quite a bit of creativity is required to combine these features in a practically useful manner.

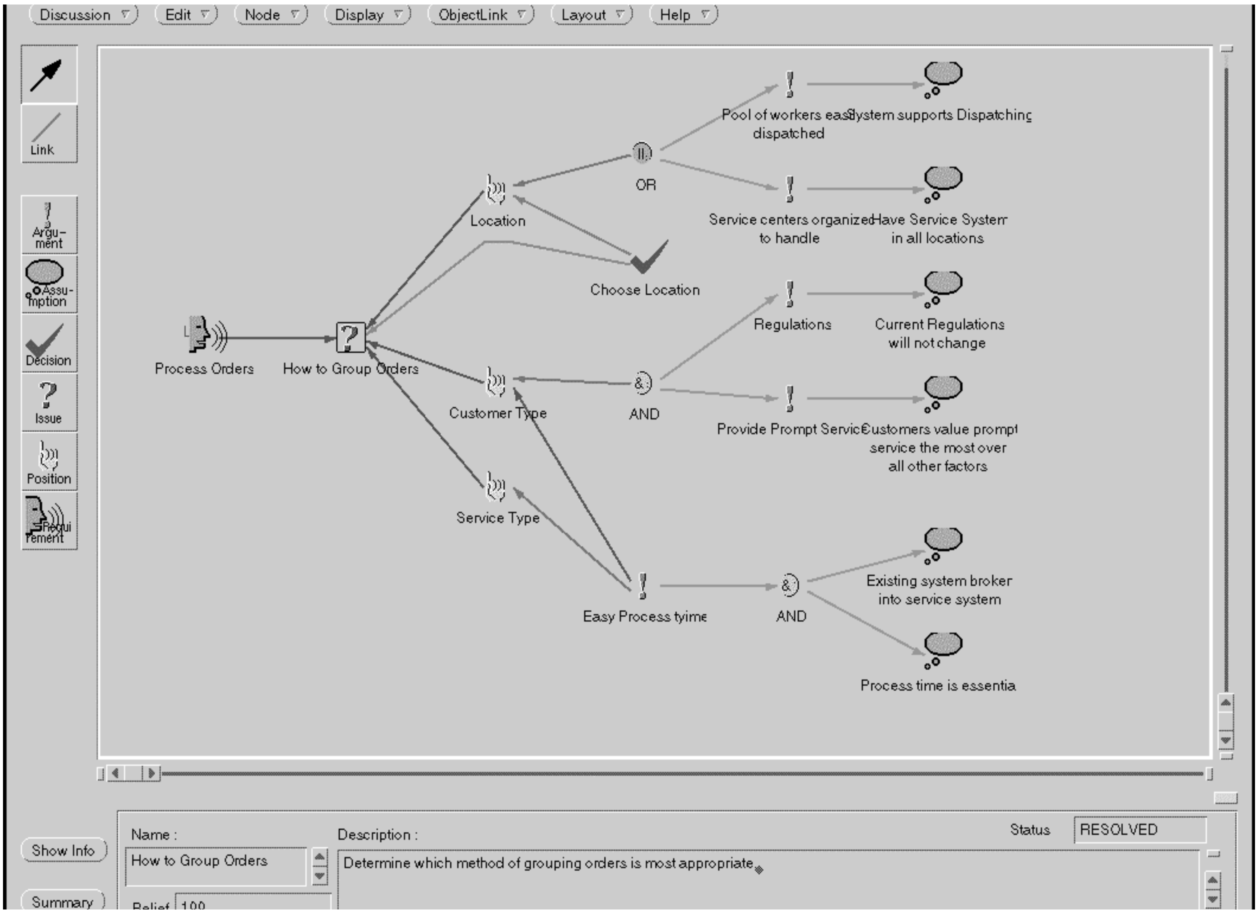


Figure 22: Rationale in Detailed View

10. Summary

We have conducted empirical studies in a broad range of software development organizations to come up with reference models for the objects and traceability links to be recorded. These models are presented at two levels of user sophisticated, which could be fairly clearly distinguished in the organizations and mark very different understandings of the importance of requirements traceability.

A simple meta model derived as part of the pre-studies also indicates how, in principle, these reference models can be extended by explicit consideration of the different stakeholders involved (contribution and usage structures, cf. (Gotel and Finkelstein 1997, Yu and Mylopoulos 1994)). The mapping of the conceptual trace objects to technical media, sources and documents – the third element in our basic meta model – was addressed by a discussion of how the different traceability link categories can be supported by inference services and what other requirements on traceability mechanisms the intended usage of the traceability link types implies. The importance of the differentiation between conceptual objects and media is also emphasized in other case studies of RE practice (Nissen et al. 1996).

In summary, we believe that the presented models constitute a good first step towards reference models for requirements traceability. One evidence for this statement is the surprisingly broad acceptance of early versions of several submodels in industry.

The accompanying longitudinal case study indicates that the uptake of these models, if accompanied by appropriate management support and user attitudes, may indeed yield substantial benefits in terms of quality as well as efficiency, leading to improved software process maturity (Ramesh et al. 1997). Of course, this case study will have to be backed up by others which would be greatly helped if our findings concerning the next generation of tools will be addressed beyond the present status by vendors. We identified structured knowledge representation, link-type related inference capabilities, the grounding of (possibly multiple) trace models in thick multi-media descriptions, and a model-driven tool-supported trace process as important ingredients. Our current efforts are devoted to maturing such tool technologies and evaluating them in industrial settings.

References

M. Alford. Strengthening the systems engineering process. *Proc. NCOSE*, San Jose, Ca, 1991.

R.C.Anderson, C. Heath, P. Luff, and T. Moran. The social and the cognitive in human-computer interaction. Technical Report EPC-91-126, Rank Xerox EuroPARC, Cambridge, U.K., 1991.

C.Baudin, C. Sivard, and M. Zweben. Recovering rationale for design changes: A knowledge-based approach. P*roceedings of the IEEE,* Los Angeles, CA, 1990.

P.A. Bernstein, T. Bergstraesser., J. Carlson, S. Pal, P. Sanders, D. Shutt. Microsoft Repository Verison 2 and the Open Information Model. *Information Systems 24*, 2 (1999).

B.J.Brown. Assurance of software quality. SEI Curriculum Model SEI-CM-7-1.1, Software Engineering Institute, Pittsburgh, Pa, 1987.

E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley 1985.

Cadre Technologies. Cadre/RQT User Manual, 1992.

J. Conklin and M.L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303--331, October 1988.

R. Conradi, B. Westfechtel: Version models for software configuration management. ACM Computing Surveys 30(2):232-282, 1998.

P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou: The Software Information Base – a server for reuse. *VLDB Journal* 4(1):1-43, 1995.

V. Dhar, M. Jarke: Learning from prototypes. *Proc. 6th Intl. Conf. Information Systems,* Indianapolis 1985, 114-133

V.Dhar, M. Jarke: Dependency-directed reasoning and learning in systems maintenance support. *IEEE Trans. Software Engineering,* 17(2), February 1988.

DoD-2167a. *Military Standard – Defense System Software Development*. US Department of Defense, 1988.

R. Dömges, K. Pohl. Adapting traceability environments to project-specific needs. *Comm. ACM 41*, 12 (1998), 54-63.

M. Dorfman, R.H. Thayer. *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press, 1990.

P. Dourish and S. Bly. Portholes: Supporting awareness in distributed work group. In *Proc ACM Conference on Human Factors in Computer Systems CHI '92*, Monterey, CA, 1992.

M. Edwards, S. Howell. A methodology for system requirements specification and traceability for large real-time complex systems. Technical report, Naval Surface Warfare Center, 1991.

H. Fellows. *The Art and Skill of Talking with People: A New Guide to Personal and Business Success*.

Prentice-Hall, Englewood Cliffs, N.J., 1964.

A. Finkelstein, J. Kramer, B. Nuseineh (eds.). *Software Process Modeling*. John Wiley RSP, 1994.

J.D. Fiksel. New requirements management software supports concurrent engineering. CimFlex Teknowledge Corporation, Washington, DC, 1994.

G. Fischer, J. Grudin, A. Lemke, R. McCall, J. Ostwald, B. Reeves, and F. Shipman. Supporting indirect collaborative design with integrated knowledge-based design environments. *Human-Computer Interaction*, 7:281--314, 1992.

C. Geertz. *Thick Description: Toward an Interpretive Theory of Culture*, chapter 3. Interpretation of Cultures. Basic Books, New York, 1973.

R. Goldman-Segall. Collaborative Virtual Communities using learning constellations: a multimedia research tool. In E. Barrett (Ed), *Sociomedia: Multimedia, Hypermedia and the Social Construction of Knowledge*. MIT press, Cambridge, Ma, 1992.

A. Finkelstein: An analysis of the requirements traceability problem. Proc. First Intl. Conf. Requirements Engineering, Colorado Springs, Co, 1994, 94-101.

O. Gotel, A.Finkelstein: Contribution structures. Proc. 2nd Intl. Symp. Requirements Engineering, York, UK, 1995, 100-107.

O. Gotel, A.Finkelstein: Extended requirements traceability – results of an industrial case study. Proc. 3rd Intl. Symp. Requirements Engineering, Annapolis, Md, 1997, 169-178.

S.Greenspan, C.McGowan. Structuring software development for reliability. *Microelectronics and Reliability* 17.

V.L.Hamilton and M.L. Beeby. Issues of traceability in integrating tools. *Proc. Colloquium IEE Professional Group C1*, London 1991.

H. Hau and R. Kashyap. Belief Combination and Propagation in a Lattice-Structured Network. *IEEE Transactions on Systems, Man, and Cybernetics*, January/February 1990.

J. R. Hauser and D. Clausing. The House of Quality. *Harvard Business Review*, May-June 1988.

C. Heath and P. Luff. *Explicating face-to-face interaction*. In G.Gilbert (Ed), *Researching Social Life*. Sage 1992.

B. Hussain, A. Ismael, M. Bender. Evidence combination using fuzzy linguistic terms in a dynamic, multisensor environment. *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems,* Las Vegas, 1994.

M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, S. Eherer. ConceptBase – a deductive object base for meta data management. *Intl. J. Intelligent Information Systems* 4(2):167-192, 1995.

M. Jarke, K. Pohl: Establishing visions in context – towards a model of requirements processes. *Proc. 14th Intl. Conf. Information Systems*, Orlando, Fl, 1993.

M. Jarke, K. Pohl, C. Rolland, J.R. Schmitt. Experience-based method evaluation and improvement: a process modeling approach. *Proc. IFIP 8.1 Working Conf. CRIS 94*, Maastricht, Netherlands, 1994, 3-28.

B. Jordon. Technology and social interaction: Notes on the achievement of authoritative knowledge in complex settings. Technical Report IRL92-0027, Institute for Research on Learning, Palo Alto, CA, 1992.

M. Kidd. Applying hypermedia to medical education: An author's perspective. *Educational and Training Technology International*, 29(2):143--151, 1992.

J. Lee. Design rationale capture and use. *AI Magazine*, 14(2), Summer 1993.

F. Lakin, J. Wambaugh, L. Leifer, D. Cannon, and C. Sivard. The electronic design notebook: Performing medium and processing medium. *International Journal of Computer Graphics*, 5:214--226, 1989.

M. Mead. *Anthropology in a Discipline of Words*, chapter 2. In Principles of Visual Anthropology, (Ed.) P. Hockings. Mouton Publishers, Paris, 1975.

T.P. Moran, J.M. Carroll (eds.). *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum 1996.

J. Mylopoulos, L. Chung and B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, June 1992.

J. Mylopoulos, M. Jarke, M. Koubarakis. Telos – a language for representing knowledge about information systems. *ACM Trans. Information Systems* 8(4):327-362, 1990.

H.W. Nissen, M.A. Jeusfeld, M.Jarke, G. Zemanek, H.Huber: Managing requirements perspectives with meta models. *IEEE Software*, March 1996, 47-58.

J. D. Palmer, Traceability, In Software Requirements Engineering, R.H. Thayer and M. Dorfman (Eds), IEEE Computer Society Press, 1997 (364:374).

F.Pinheiro and J.Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, March 1996.

K.Pohl: The three dimensions of requirements engineering: a framework and its applications. *Information Systems* 19(3):243-258, 1994.

K.Pohl. *Process-Centered Requirements Engineering*. John Wiley Research Science Press, 1996.

K. Pohl, K. Weidenhaupt, R. Dömges, P. Haumer, R. Klamma, M. Jarke. Process-Integrated Modeling Environments (PRIME): Foundations and Implementation Framework. To appear, *ACM Transactions on Software Engineering Management*, 1999.

QSS Ltd. Dynamic Object Oriented Requirements System, Reference Manual, Oxford, UK, 1996.

B. Ramesh. Factors influencing requirements traceability practice. *Comm. ACM 41*, 12 (1998), 37-44.

B.Ramesh and V. Dhar. Supporting systems development using knowledge captured during requirements engineering. *IEEE Transactions on Software Engineering*, June 1992.

B.Ramesh and M. Edwards. Issues in the development of a model of requirements traceability. *Proc. International Symposium on Requirements*, San Diego, CA, January 1993.

B.Ramesh, C. Stubbs, T. Powers, and M. Edwards. Requirements traceability – theory and practice. *Annals of Software Engineering* vol. 9, 1997 (forthcoming).

J.Rochelle, R. Pea, and R. Trigg. Videonoter: A tool for exploratory video analysis. Technical Report IRL90-0021, Institute for Research on Learning, Palo Alto, CA, 1990.

W.H.Roetzheim. *Developing Software to Government Standards*. Prentice Hall, 1991.

T.Rose, M.Jarke, M.Gocek, C.G.Maltzahn, H.W.Nissen. A decision-based configuration process environment. *IEE Software Engineering Journal* 6(5):332-346, 1991.

Ryle. *Concept of Mind*. Harmonds Worth, 1949.

A.W.Scheer: *Reference Models for Business Processes.* Springer 1995.

H. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 1981.

S.Shum and N. Hammond. Argumentation-based design rationale: From conceptual roots to current use. Technical Report EPC-93-106, Rank Xerox Limited, Cambridge EuroPARC, Cambridge, CB2 1AB, UK, 1993.

R.Smithers, M.Tang, N.Tomes. The maintenance of design history in AI-based design. *Proc. Colloquium by IEE Professional Group C1*, London 1991.

R.Stallman, G.Sussman: Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9(2):135-196, 1977.

G.Stehle. Requirements traceability for real-time systems. *Proc. EuroCASE II*, London 1990.

R.Stults. Experimental uses of videotapes to support design activities. Technical Report SS 1-89-19, Xerox Palo Alto Research Center, Palo Alto, CA, 1988.

TD Technologies. *SLATE User Manual*. Dallas, Tx, 1996.

J.F. Templeton. Focus Groups: A Guide for Marketing and Advertising Professionals, Probus Publishing Company, NY, 1987.

P. Winston. Artificial Intelligence. Addison-Wesley, Reading, MA, 1984.

S.Wright. Requirements traceability - What? Why? And how? *Proc. Colloquium IEE Professional Group C1*, London, UK, 1991.

E. Yu, J. Mylopoulos: Understanding 'why' in software process modeling. Proc. 16th Intl. Conf. Software Eng., Sorrento, Italy, 1994, 159-168.54