A DECISION RATIONALE MANAGEMENT SYSTEM:

CAPTURING, REUSING, AND MANAGING THE REASONS FOR DECISIONS

by
Jintae Lee

B.A. Mathematics, Univ. of Chicago (1979)
M.Phil. Univ. of Cambridge (1982)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE
at the
MASSACHUSETTS INSITITUTE OF TECHNOLOGY
February 1992

Signature of Author _____
Department of EECS
September 20, 1991

Certified by _____
Patrick H. Winston
Professor, Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by _____
Campbell L. Searle
Departmental Committee on Graduate Studies

A Decision Rationale Management System: Capturing, Reusing, and Managing the Reasons for Decisions

by
Jintae Lee

Submitted to the EECS Department on September 17, 1991,
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science

Abstract

This thesis identifies the needs for capturing and managing decision rationales, articulates the concept of a decision rationale management system that meet these needs, and presents a computer system that implements the concept.

Capturing and managing decision rationales, i.e. the deliberations leading to decisions, can bring about many benefits. The rationales can then be used to support decision making, can be shared among decision makers, and can be reused for similar decisions. A decision rationale management system, i.e. a system that captures and manages decision rationales to provide these benefits, requires three major components: a language for representing elements of rationales, a method of using the language to capture the rationales, and a set of services that use the captured rationales to support decision making.

This thesis articulates a model of decision rationales and uses it to develop DRL, a language for representing the elements in this model. The thesis also presents SIBYL, a computer system that helps people to capture rationales in DRL by providing a number of interface features intended to reduce the overhead associated with explicit representations. Using the rationales captured in DRL, SIBYL provides computational decision services, such as retrieving useful rationales from past decisions, maintaining dependencies among the various elements of rationales, and keeping track of multiple decision states. These services realize the benefits of a structured representation of rationales, and provide further motivation for capturing rationales in DRL.

Thesis Supervisor: Patrick H. Winston
Title: Professor

Acknowledgements

First, I thank the members of my thesis committee, Professors Patrick H. Winston, Thomas W. Malone, Randall Davis, and Marvin Minsky, for their guidance and patience. Patrick has supplied the enthusiasm for this research that I needed all along. Tom has brainwashed me with his ideas that are now difficult to shake off. Randy has been the source of constant critique, which undoubtedly made the research more painful but better. Marvin has supplied me with his vision and pointers that only he could have given.

I thank all my friends at the AI Laboratory and the Center for Coordination Science: Kum-Yew Lai, Lukas Reucker, Rick Lathrop, Kevin Crowston, Franklyn Turbak, Paul Resnick, John Mallery, Roger Hurwitz, Gary Borchardt, Jolene Galegher, Mark Ackermann, and many others. Kum-Yew -- with his interests ranging from artificial intelligence, human computer interaction, finance, civil engineering, computer-supported cooperative work, systems dynamics, economics to mention only a few -- was a friend who couldn't say no and whom I could always relied on for bouncing out ideas. He has been sorely missed ever since he left MIT to become a student of finance. I hold him partially responsible for any error or fuzziness in this thesis because if he stayed around, he would have undoubtedly pointed them out to me. Lukas -- amid the thousand activities that he somehow juggles, ranging from parachuting, scuba diving, organizing lab tours, lecture series committee, Anemie club, to mention a few -- provided me with the most detailed comments on this thesis than anybody else. His comments made reading this thesis less painful than would be otherwise. Rick has also been an invaluable source of encouragement and ideas when I was down and out. I guess I could write a book detailing my appreciations.

I would like to thank "design rationalizers": Tom Moran, Jack Carroll, Jeff Conklin, Ray McCall, Gerhard Fischer, Allan MacClean, Simon Shum, Mark Novic, and others who helped me directly or indirectly see better the issues involved in managing rationales. I would also like to thank people at Xerox PARC and GTE Laboratory: Frank Halasz, Cathy Marshall, Susan Newman, Austin Henderson, Danny Bobrow, Frank Manola, Michael Brodie, and many others that I talked to in these places. Working at these places not only saved me and my family from starving, but also was a wonderful way to work out and bounce off ideas. There are others who came across my path and left their marks in this research. Randy Trigg, despite our brief interaction, comes to my mind, but I am sure there are many many more. I apologize for not being able to thank them individually, but that is only because I am running out of space and time.

At last, but most of all, I thank my family, Hyokyung, Youngwon, and Jangwon, who always greeted me with smiles and made me smile even during my darkest hours. And my parents, who have been very patient with my progress and always supportive. And my parents-in-law who put their daughter in the trust of a man who earns minimum wages and yet never complained. I thank them all from the deepest of my heart.

Contents

1. Introduction 1

1.1 The Problems 4

1.2 Summary of Approaches 9

2. A Scenario 14

3. The Structure of Decision Rationales 30

3.1 What do we want to do with decision rationales? 31

3.2 Models of Decision Rationale 32

4. DRL: A Decision Representation Language 41

4.1 Overview 42

4.2. DRL's Representation of the Decision Rationale Model 46

5. SIBYL: An Environment for Using DRL 52

5.1 Overview 53

5.2 Implementation 57

5.3 Setting up an Initial Structure 58

5.4 Releasing the Initial Structure 64

5.5 Augmenting the Decision Structure 65

5.6 Making the Decision 75

5.7 Evaluating the Outcome of the Decision 76

6. Computational Services 79

6.1 Precedent Management 81

6.1.1 Specific Retrieval Request 81

6.1.2 General Retrieval Request 85

6.2 Dependency Management 104

6.2.1 The Scope 104

6.2.2 Implementation I 107

6.2.3 Implementation II 111

6.3 Viewpoint Management 113

6.3.1 Representation 114

6.3.2 User Interface 116

6.3.3 Implementation 120

6.4 Other Services 121

7. Comparison to Related Work 128

7.1 Systems that Capture Rationales 129

7.2 Semi-Formal Rationale Management Systems 134

8. Conclusion 148

8.1 Contributions 148

8.2 Future Research 152

Appendix: Details of DRL 157

References 177

Chapter 1

Introduction

A structured representation of decision rationales, i.e. the deliberations leading to a decision, can bring about many benefits. The knowledge that decision makers bring to the decision becomes available for other people or computational agents to share, augment, and argue about. The representation of a decision making process serves as a document of how the decision developed, which in turn can serve as a basis for learning and justification. In addition, a well-structured representation provides a basis for defining computational services for decision making, such as keeping track of dependencies and retrieving useful rationales from past decisions.

Some of these benefits have been explored so far by a few systems, but most of these systems require highly formalized and structured domain knowledge. For example, the

research on derivational analogy systems [Carbonell 1986; Huhns & Acosta 1988; Mostow 1989; Steinberg & Mitchell 1985] explore ways of capturing the reasoning behind design and reusing parts of the reasoning trace for solving redesign or similar design problems. These systems, however, require that the domain knowledge be sufficiently formalized so that the problem solver can run in the first place. As a result, they are inapplicable to domains where the knowledge is less structured, not well understood yet, or too expensive to be formally represented. Even in the domains (e.g. design of circuits or parsers) whose knowledge has been formalized, decisions typically involve consideration of other worldly knowledge -- such as availability of resources or political influences -- which may be difficult to formalize. Nevertheless, capturing and managing rationales is no less important or urgent in these domains.

The goal of this thesis is to develop a system that captures and manages rationales in such a way that their benefits can be realized without requiring the formalized domain knowledge. This goal is achieved by developing a representation that allows informal description of domain knowledge to be included in formal structures. These formal structures need to capture generic knowledge about decision making, and support interesting computational operations. Because people are still necessary to interpret the informal descriptions, any system that uses this type of representation, i. e. semi-formal representation [Lai et al. 1988], requires much human interaction. Nevertheless, it helps solve the problem of brittle performance because people can supply the expertise or commonsense where the system cannot, at least until we gradually understand more of the domain and make the system understand it too.

The last point about gradual understanding leads to another goal of this research, that of producing a system that is practically useful. Apart from its obvious merit, this goal is derived from the more ambitious goal of producing an automated rationale management

system. Although informal descriptions provide us with flexibility, they need to be formalized if one's goal is to automate rationale management as much as possible. This goal of automation is similar to the one underlying the derivational analogy systems mentioned above, but this research takes the approach of incremental formalization in realizing this goal. The approach is to start with a system that is useful even without much understanding of the domain knowledge, but have the system record the domain knowledge in the course of its use, which can then be formalized and fed back into the system, thereby making it more powerful. This feedback loop is possible in a rationale management system because the rationales captured by the system embody knowledge about the domain, though it might be in the form of informal descriptions. Thus, a system that captures and manages rationales also helps us gain more understanding of the domain and formalize the knowledge. For this approach to work, however, it is important that the system is useful enough to be used in real situations. Although the usefulness of the current system cannot be claimed without much qualification, the attempt to make a useful system has generated many constraints and has been achieved to a degree, as discussed in the thesis.

The thesis proceeds as follows. In the rest of this chapter, I first describe and categorize a number of concrete problems that motivated this research (Section 1.1). I then discuss the approach that this thesis takes to solve these problems (Section 1.2). In particular, I propose the concept of a decision rationale management system as a solution to these problems and articulate its components. This discussion provides a preview of the thesis. In Chapter 2, a scenario illustrates concrete behaviors that I believe an ideal decision rationale management system should have. This scenario serves as a yardstick against which the success of this thesis is measured.

The next four chapters constitute the main body of this thesis. Chapter 3 identifies the elements of a decision rationale that need to be represented by building a sequence of models which differentiate, in different degrees, the internal structure of decision rationales. These models are also used as a framework for later discussions. Based on these models, Chapter 4 presents a rationale representation language, called DRL (Decision Representation Language). Chapters 5 and 6 present SIBYL, the decision rationale management system that use DRL to capture and manage rationales. Chapter 5 describes the environment that SIBYL provides for using DRL to capture rationales. Chapter 6 discusses the computational services that SIBYL provides by using the captured rationales. These three chapters, 4, 5, and 6, describe each of the components of a rationale management system: language, method, and services.

Chapter 7 compares SIBYL to other tools that support decision making. First, the existing rationale management systems are located along two dimensions: the formalization required and the reusability of the rationales (Section 7.1). Then, the tools that are closest to SIBYL in this space are discussed in detail (Section 7.2). Chapter 8 concludes the thesis by summarizing its contributions and discussing the topics for future research.

1.1 The Problems

This research grew out of my experience in helping a group to set up its computing environment. We had to decide, for example, which workstation to buy, which network protocol to use, and whether to use a relational database or an object-oriented database. One of the most frustrating aspect of this experience was the realization that although hundreds of groups must have made similar decisions in the past, we had to start from

scratch because the information that must have been collected and analyzed in these decisions, i.e. their decision rationales, were not available to us. Likewise, hundreds of groups would make similar decisions in the future, and they would not be able to benefit from our experience because they would not have access to our decision rationale. The difficulty in reusing rationales is only one of the problems that we face in decision making. Other problems include keeping track of the issues discussed and those yet to be resolved, explaining the rationale to other groups, or even just physically getting people together to talk about these issues.

These problems can be categorized as follows. Consider a decision making process as consisting of sessions, e.g. meetings in case of group decision making or individual deliberations otherwise (Fig. 1.1). First, there are problems of *managing rationales within a session* like being able to find out what depends on what or ask "what-if" questions. Then there are problems of *managing rationales across sessions* -- for example, remembering what have been resolved in previous sessions and what need yet to be resolved. Across decisions, there are problems of *sharing rationales across decisions* taking place concurrently, such as sharing information among groups making similar decisions. Finally, also across decisions but separated in time, there are the problems of *reusing rationales from past decisions*, as discussed above. Each of these categories is described in the rest of this section.

Managing Rationales within a Session

In making decisions, we often need to keep track of dependencies, compare multiple viewpoints, and ask "what-if" questions. There are many techniques and tools that address the problems in this category -- for example, decision analysis and simulation tools. In

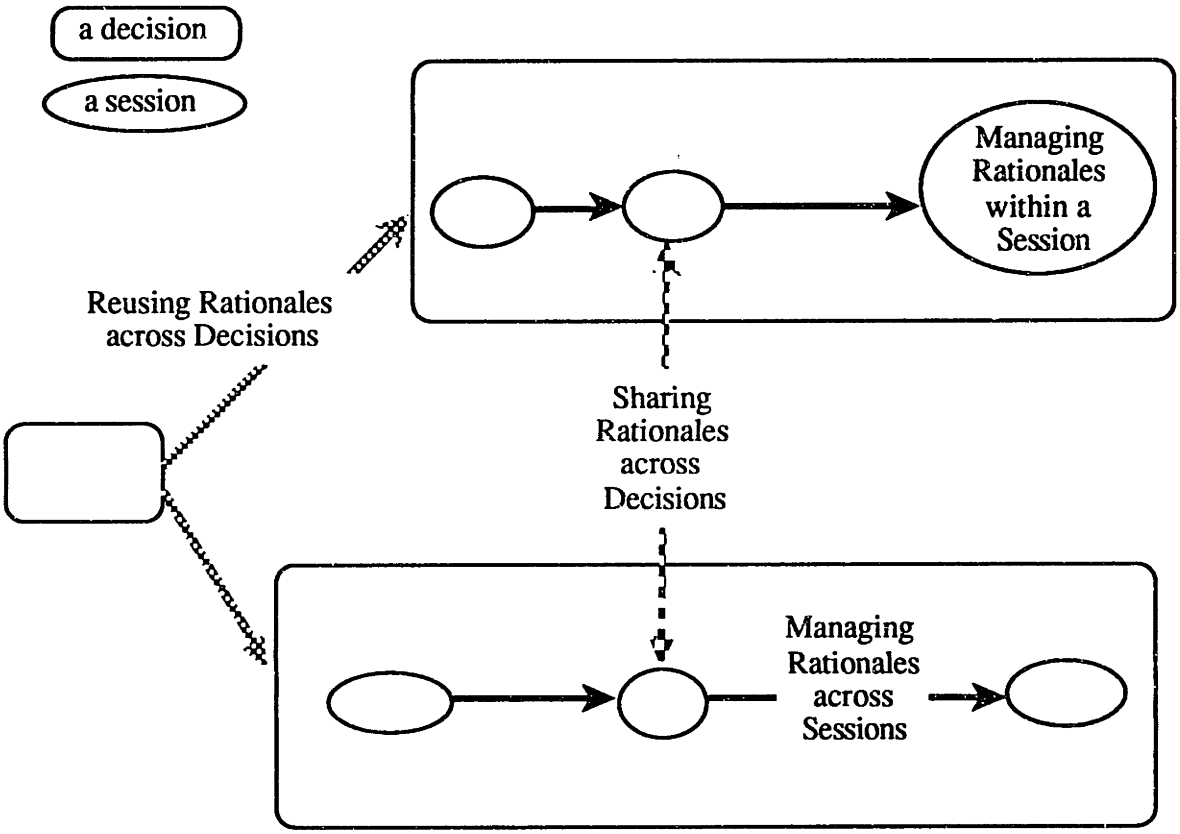


Figure 1.1 Kinds of needs for managing rationales

many decisions, however, these tools are too rigid for use. For example, we could not use these tools in our decisions about the database or the network protocol because the use of tools required formalization of a wide variety of concerns that were either too difficult or too expensive to formalize. Furthermore, these tools are useful only after we have a clear understanding of the factors that influence our decisions; they do not help us find out what these factors are.

Managing Rationales across Sessions

Different kinds of problems arise in managing rationales across sessions within a decision making process. Here, typical problems include those of bookkeeping, for example remembering the issues discussed previously, the issues yet to be resolved, or what is being done about them. Associate with this category is the problem of ensuring that the individuals involved in decision making can exchange their ideas in a quick and focused manner. Typically, decision making takes place in face-to-face meetings, which necessitates finding a place and a time in which the decision makers can all meet together. This problem, as many people complain, often unnecessarily prolong the decision making process. In addition, face-to-face meetings have other known problems. For example, people have to wait until the meeting to contribute their knowledge, or may not contribute their ideas if meetings are dominated by certain individuals

Sharing Rationales across Decisions

Often, the different subgroups need to communicate their rationales to other groups so that the groups reach a common understanding of the problems and share their own expertise. For example, in designing a product, designers might want to know why the marketing group wants some feature and dislike others. Another problem in this category arises when two groups making similar decisions want to share information. For example, at one point I worked for two groups making similar decisions about hardware and software platforms. The two groups shared many requirements and alternatives, hence were in the position of benefiting from shared information. By virtue of being involved in both, I was able to transfer some knowledge but this transfer was accidental and could have been more systematic.

Reusing Rationales from Past Decisions

We do not reuse enough knowledge from past decisions. We often make decisions similar to the ones that we or other groups have made before. For example, I have helped three more groups make decisions about the computing environment since the original experience that I referred to. There were enough requirements and alternatives (e.g. minimize cost, interface to email) that were shared among these groups so that many pieces of the knowledge accumulated by one group could have been useful to the others.

The reusable knowledge includes more than factual information; the knowledge from past decisions might reveal a critical requirement that current decision makers did not think about, an option they were unaware of, an assumption they mistakenly held, or an argument they did not consider. Also valuable is the knowledge about how different requirements relate to one another; for example, the knowledge that minimizing cost can typically be factored into minimizing purchase cost, minimizing maintenance cost, and minimizing development cost. If nothing else, the knowledge of the requirements considered in similar past decisions can serve as a useful checklist. Much of the useful knowledge from past decisions, however, is usually ignored. As a result, we often repeat the same efforts and mistakes.

Moreover, we need to understand the rationales for a past decision for reasons other than reusing them in similar decisions. Another problem in this category is that of keeping an audit trail so that we can justify or review our decisions. Yet another problem is in design, where the rationales for the original design can be valuable for troubleshooting or redesign of the artifact. In all of these cases, if we had some way of selectively accessing the relevant parts of the deliberations underlying past decisions, we would be able to learn

much from them, not to mention saving the effort of collecting those pieces of knowledge in the first place.

1.2 Summary of Approaches

In the last section, I described and categorized the problems that motivated our research. I propose a Decision Rational Management System as a solution to these problems. A Decision Rationale Management System (RMS) has three components:

(1) a *language* for describing the elements of decision rationale,

(2) a *method* for using the language to capture the rationales,

(3) a set of *services* that use the captured rationales to provide decision support.

For example, a simple example of decision rationale management system is to

(1) use English,

(2) write down everything said by anybody in the decision making process, and

(3) find whatever we need from the record by brute force (e.g. by flipping through pages of the notebooks used).

This solution is in most cases unsatisfactory. The cost of writing down everything and later finding what is needed is so large that the benefit of actually finding something is not likely to override the cost.

So we need to design a decision rationale management system, i.e. a tool for supporting decision rationale management, whose benefits exceed its costs. In other words, the challenge in the design of a decision rationale management system is to:

(1) design a language that is structured enough to capture the *important* elements of design rationale and their relations,

(2) develop a method *which helps reduce the cost of using the language to capture rationales,* and

(3) define services *which reward the user for recording rationales.*

A rational management system (RMS)[1] that satisfies these constraints can solve or alleviate the problems discussed in the previous section. This thesis substantiate this claim by developing a system called SIBYL. Its language, DRL, is based on a model that characterize the important elements of decision rationales (Chapter 3), and provides constructs for representing them (Chapter 4). SIBYL provides an environment in which to capture rationales in DRL as a by-product of making decisions. This environment is built on top of Object Lens [Lai et al. 1988], a tool for building computer supported cooperative work, and uses its features such as template editors and various display formats, to make the rationale capture easier (Chapter 5). SIBYL also provides a set of computational services, such as keeping track of dependencies, comparing multiple versions, and retrieving relevant rationales from past decisions, that reward the user for recording rationales (Chapter 6). In the rest of this section, I provide an overview of how SIBYL addresses the problem categories discussed above.

Managing Rationales within a Session

As mentioned in the previous section, there are many decision support tools, such as decision analysis, help manage rationales, but their use requires the precision or the formalization often not available or too expensive to produce in many decisions. In order

[1] In the rest of the thesis, I will often abbreviate Decision Rationale Management System as Rationale Management System.

to manage rationales in these situations, SIBYL implements DRL as a semi-formal representation (Chapter 3). That is, the constructs of DRL for representing the elements of rationales, such as alternatives, goals, and arguments about them, are implemented as formal types with their own attributes, but allows the values of these attributes to be a mixture of formal and informal descriptions. This way, the domain knowledge can be entirely described informally (as attribute values of the formal constructs), but SIBYL can still help manage rationales based on the way that the formal constructs have been used. For example, all the information, say about an interaction manager, may be given in English, but SIBYL can provide its service as long as it knows that it is an alternative, one of the DRL constructs because it knows how an alternative should relate to other constructs of DRL.

Using the rationales captured in DRL, SIBYL can provide the following services for managing rationales within a session. SIBYL allows the user to maintain the consistency among the elements of rationales (Section 6.2). For example, when an assumption (represented as a claim in DRL) is no longer true, the user can ask SIBYL to invalidate the arguments that depend on it. The user can also create and compare multiple decision states, for example, a given decision under different assumptions (Section 6.3). Using the structure of DRL, SIBYL can also keep track of the rationales by collecting and displaying, for example, all the arguments evaluating a given alternative, or the criteria used for the evaluations. These services are not as powerful as those provided by some other decision support tools and often require interaction with users in the absence of formalized domain knowledge. On the other hand, SIBYL allows the user to create formal objects modeling domain knowledge, and write rules that exploit the knowledge to obtain more powerful services.

Managing Rationales across Sessions

Because the rationales are captured in DRL during a session with SIBYL, managing rationales across sessions becomes easier for several reasons. First, the rationales are permanently available in the electronic form accessible to the computer. Furthermore, because they are represented in a structured way, SIBYL can easily find decisions that are unresolved from the previous sessions, questions that need yet to be answered, or the arguments that need yet to be evaluated. SIBYL in fact helps managing rationales across sessions by making the boundary between the sessions less rigid. Because SIBYL provides a forum in which to examine and update the rationales, its users can make a decision without having to be physically together (Chapter 5). This ability to make decisions asynchronously also solves some of the problems in face-to-face meetings mentioned above: such as delay caused by meeting arrangements.

Reusing Rationales across Decisions

In addition to the benefits of providing a permanent, electronic, and structured record of the past decision rationales, SIBYL provides services that specifically help the user to retrieve relevant parts of the rationales from past decisions. There are two ways of retrieving the rationales once they are represented in DRL. If the user is looking for something specific, e.g. arguments about a given alternative or answers to a question, then the user can specify a partial structure as a query. A rule system takes this query, and retrieves any structure from past decisions that matches this partial specification (Section 6.1.1). Users, however, are often interested in relevant rationales without necessarily looking for anything specific. To support such cases, I have also developed an interactive algorithm for retrieving relevant

rationales for a given decision (Section 6.1.2). This algorithm, labelled the precedent manager, is based on the intuition that two decisions are similar to the extent that they share similar goals, and uses the goals shared between a given decision and past decisions to judge potentially relevant rationales.

Sharing Rationales across Decisions

Sharing rationales becomes easier again because DRL makes the elements of rationales and their relations explicit. For example, if the user is interested in all the arguments about an alternative or answers to a question, it is easy to collect the relevant elements and send them to the group. Using the features of the underlying Object Lens, mentioned above, SIBYL can send and receive a collection of structured objects through email or in a file, and link them appropriately to the objects that already exist in the current environment (Chapter 5). This way, a group can request and access parts of rationales of the other groups and use this knowledge as a basis for mutual understanding, for collaboration, or for further communication across groups.

In the next chapter, I describe a scenario that illustrates the overall idea from the user's point of view. This scenario illustrates the benefits that will motivate the user to use the language and the kinds of services that an RMS needs to provide.

Chapter 2

A Scenario

This chapter illustrates the behaviors of an idealized decision rationale management system which the implemented system has tried to approximate. This scenario represents a first step taken, logically as well as historically, in the present research. Logically, it specifies the goals of the research in concrete forms. Historically, the rest of the research discussed in later chapters -- design and implementation of the language, the interface, and the services -- follows the articulation of this scenario. As such, the scenario provides a way to present the goals and main ideas clearly without getting into the details of the actual interface. It also serves as a yardstick against which the success of the current research is to be measured.

The system presented below is different from the implemented system in the following ways. First,it uses natural language interface in order to get the main ideas across more clearly. The use of natural language, though beyond the capability of the implemented system, makes it possible to describe the user request and the system response without having to describe at this point the details of the SIBYL interface, such as the menus and buttons that the user needs to activate. The actual interface of the implemented system is described in Chapters 5 and 6. Also, in order to present the main goals and ideas concisely at this point in the thesis, a step in the scenario sometimes represents a number of steps in the implemented system. The scenario uses a few terms, such as issue and subissue, that are slightly different from the ones actually implemented but more intuitively obvious. A few features, such as multiple shading of the nodes in a graph, are presented in the scenario, though not implemented, when they help getting across main ideas better. The scenario therefore should be regarded as a concrete illustration of the goals and the main ideas that have guided the present research not as a description of the current system.

Imagine a group designing a window manager, i.e. the software which provides and manages windows for different applications. Typically a window manager would provide functions that an application can call upon, for example, to do something like create, move, resize, and scroll a window or structure the components of a window in a certain way.[2] There are different ways in which these functions can be provided to the applications. At

[2] Window manager design was chosen as an example domain for several reasons. First, it is a domain that we are just beginning to understand. The domain knowledge has not been formalized, but parts of it could be, e.g. knowledge about modularization. Thus, it provides an opportunity to explore the feasibility and the usefulness of a rationale management system in realistic situations. Also, many issues that this domain faces are potentially reusable for other domains. For example, the issue of whether to have a strict interface between modules is quite general and arise in many other domains,such as the design of operating systems or programming language. Thus, it provides a chance to explore cross-domain reusability of the rationales as well, which is not discussed in this thesis, but is a topic of future research. In constructing the scenario, [Lane 1990] and [Hopgood et al. 1985] have been very useful.

one extreme, applications can have access any of the window manager functions. At the other extreme, applications can have no access to any of the window manager functions directly but have to go through a layer of abstraction (i.e. an interaction manager or the user interface management system), which typically consists of higher level routines composed of window manager functions. There are also intermediate solutions, such as allowing direct access to the window manager functions but also providing a toolbox (or library) of higher level routines that applications are encouraged to use.

In the scenario, two alternatives are currently being considered: the interaction manager and the toolbox interfaces. These alternatives are described further in the scenario itself. The scenario starts at a point where the group has used SIBYL to raise a number of issues and accumulated some arguments concerning them. Imagine that John, a member of this group, is about to use SIBYL to catch up on the current state of the decision making and to contribute additional knowledge.

Notation:

`User's requests and the system responses are in the courier font.`

`The actions taken by the user or the system are in italic courier.`

The comments and the figure captions are in Times.

Getting Background Knowledge

User: Show me the current issues.

Target System: *displays graphically the issues and their relations. (Fig. 2.1)*

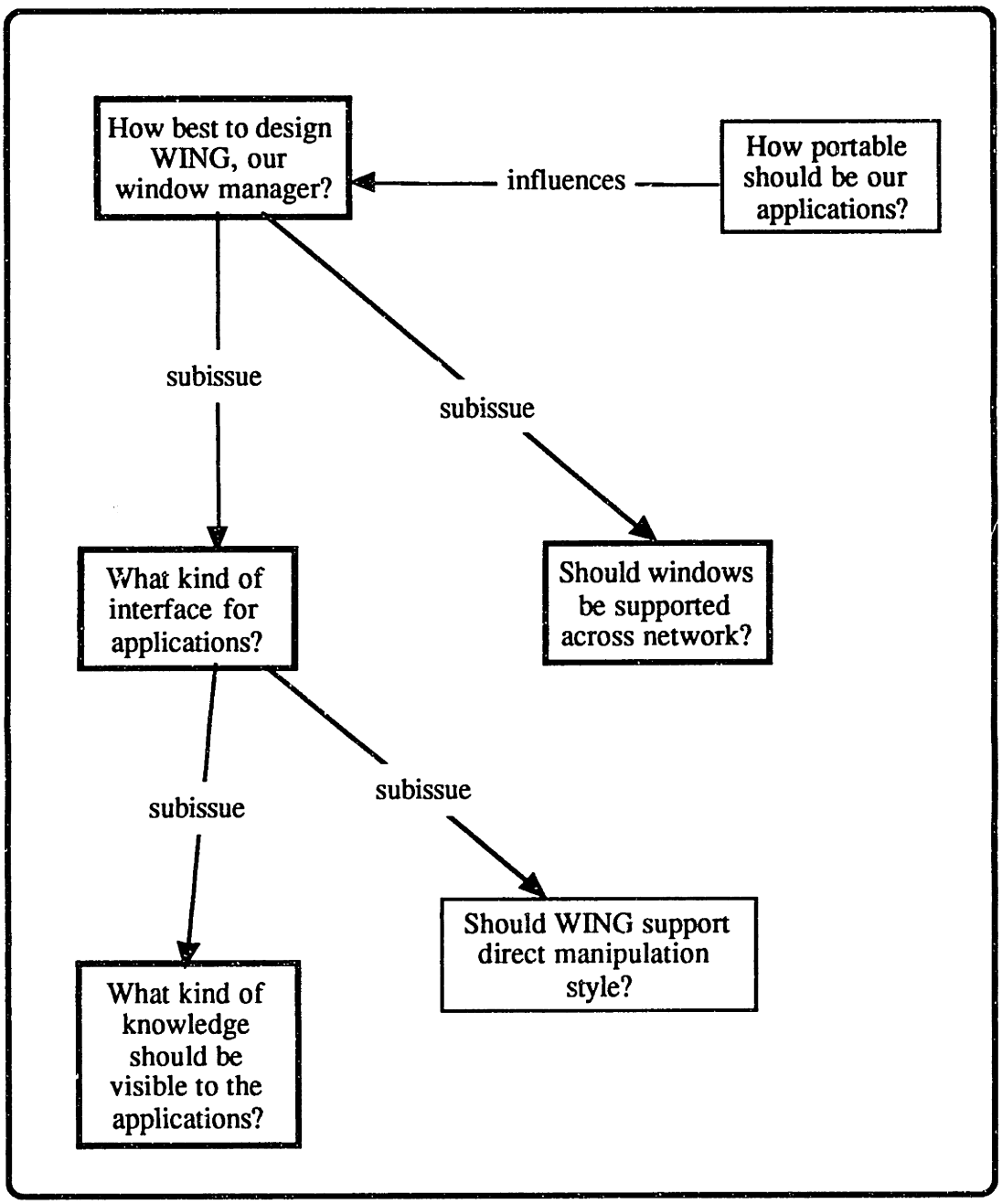


Figure 2.1 A browser displaying the issues currently being considered. The ones in heavier boxes represent unresolved issues.

User: *double mouse-clicks on the issue, "What kind of application interface?" to get more information about it.*

Target System: *displays all the information about the chosen issue in the form of attribute/value pairs. (Fig. 2.2) There are two alternatives being considered: Interaction Manager and Toolbox approach, as shown in the Alternatives attribute.*

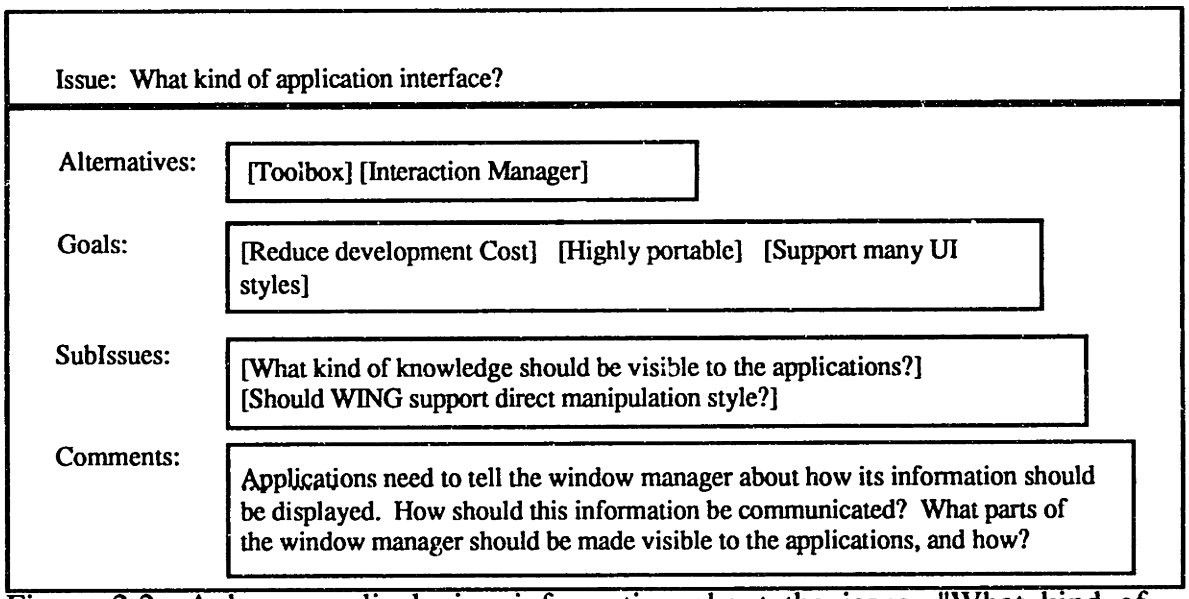


Figure 2.2 A browser displaying information about the issue, "What kind of application interface?" A square bracket denotes an object, which can be expanded and examined in more detail.

User: *double mouse-clicks on the alternatives in the Alternatives field to examine them in more detail.*

Target System: *displays the information about the two alternatives. (Figure 2.3)*

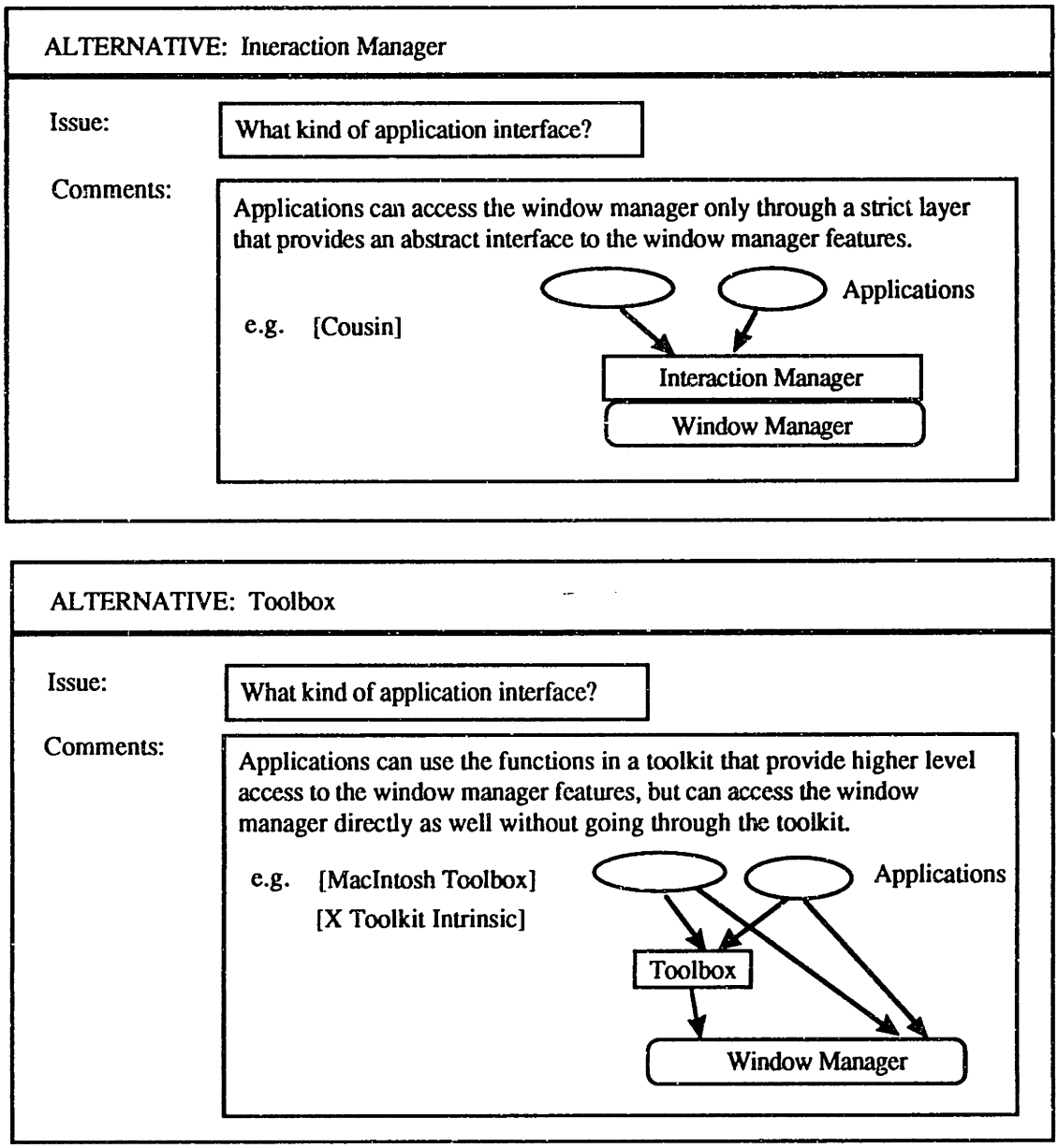


Figure 2.3 Browsers displaying two alternatives being considered in the issue, "What kind of application interface?"

Catching Up

John wants to know about the changes since the last time he looked at this decision.

The default method for displaying the changes is to show only the evaluations updated

as a result of these changes. From these evaluations, the user can trace the changes responsible for these updates, as illustrated in the scenario below. Optionally, however, the system can also replay the changes in the chronological order; each change can be displayed, for example, at the user's mouse-click.

User: Give me the updates.

Target System: The following describes the objects added or changed since the last time you used the system that might be relevant to the current decision. You might want to examine them before you proceed.

- The decision problem, [Should the windows be supported across network?], which is a subdecision of the current one, has been resolved in favor of the alternative, [YES, but only local network].

- The decision problem, [What kind of knowledge should be visible to applications?], might be worth reexamining because an assumption on which it is based has changed.

The system can be told to monitor a particular change and, when the change occurs, do certain things, like propagating its consequences and/or notifying the user .

- The price of the X terminal has just been reduced. That changes the evaluations in the decision problem, [Should windows be supported across network?]

User: Show me the current status of the issue, "What kind of application interface?"

Target System: *displays a matrix that shows current evaluations of the alternatives with respect to different criteria. (Figure 2.4)*

The Current Status of Decision Problem: What kind of application interface?			
Requirements / Alternatives	Reduce development cost	Highly portable	Implement direct manipulation
Interaction Manager	L*?#	H	M
Toolbox	M	M*	unevaluated

Figure 2.4 A decision matrix displaying the current status of the issue, "What kind of application interface?"

In this matrix, each of the alternatives being considered is evaluated with respect to the requirements that the group wants to satisfy. The requirements are shown in the columns, the alternatives in the rows, and the evaluation of the alternatives with respect to a criterion in each of the corresponding cells. The cells changed from the last examination are marked with *. The previous version is available for comparison so that we can see the new one side-by-side with the old one, Also, those evaluations for which there are unanswered questions are marked with ?, and those for which there are qualifications with # .

User: Show me the reasons for the evaluation of Interaction Manager with respect to Reduce development cost.

Target System: *displays a browser displaying arguments responsible for the evaluation.* ***(Figure 2.5)***

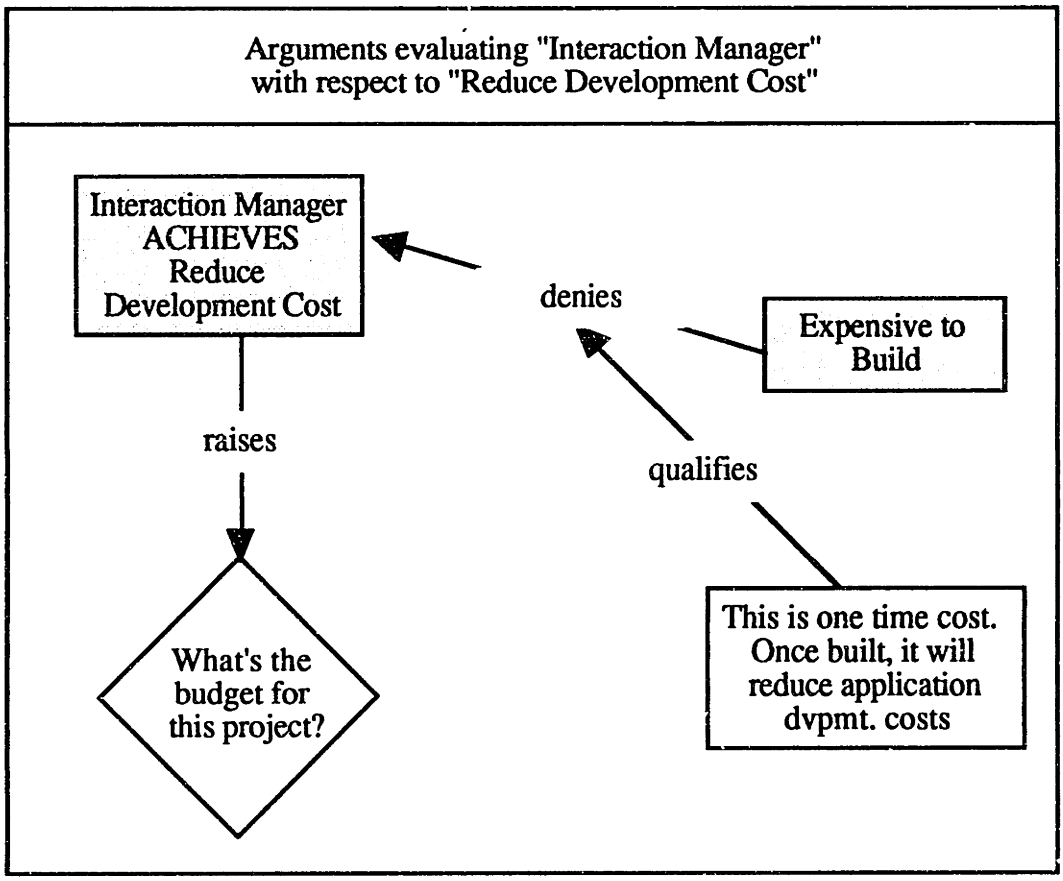


Figure 2.5 An argument browser displaying arguments that led to the current evaluation of the alternative, Interaction Manager, with respect to the criterion, Reduce Development Cost.

The arguments shown above are the result of the initial structure being augmented by the members of the decision making process.. In principle, the group members may be computational agents who have been working on finding relevant decisions, although

the current implementation does not support such a feature. The nodes shaded are the ones that have remained unchanged.

The user could also find out the reasons underlying other evaluations by examining the argument browser associated with each of the evaluations. Also, if the user does not understand some of the arguments, the user can get more information by expanding the nodes in question. For example, suppose that John did not understand what the qualification was in the above argument browser.

User: double clicks on the qualifying claim to expand and examine it in more detail.

Target System: displays the qualifying claim in full. (Fig. 2.6)

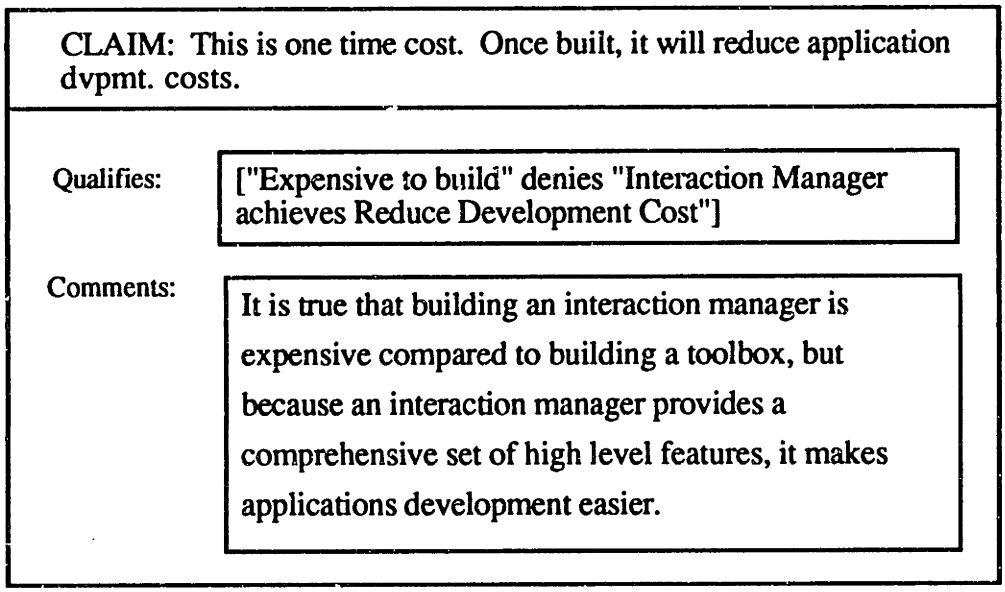


Figure 2.6 A browser displaying the claim qualifying another claim that "expensive to build" denies "Interaction Manager achieves Reduce Development Cost." Note that this claim being qualified is shown as a link labelled "denies" in Figure 2.5.

Making Changes

User: *enters additional arguments, and replies to question raised after examining the current state of the decision,*

Target System: *helps the user to update the argument browser through context sensitive menus and displays the updated browser (Fig. 2.7)*

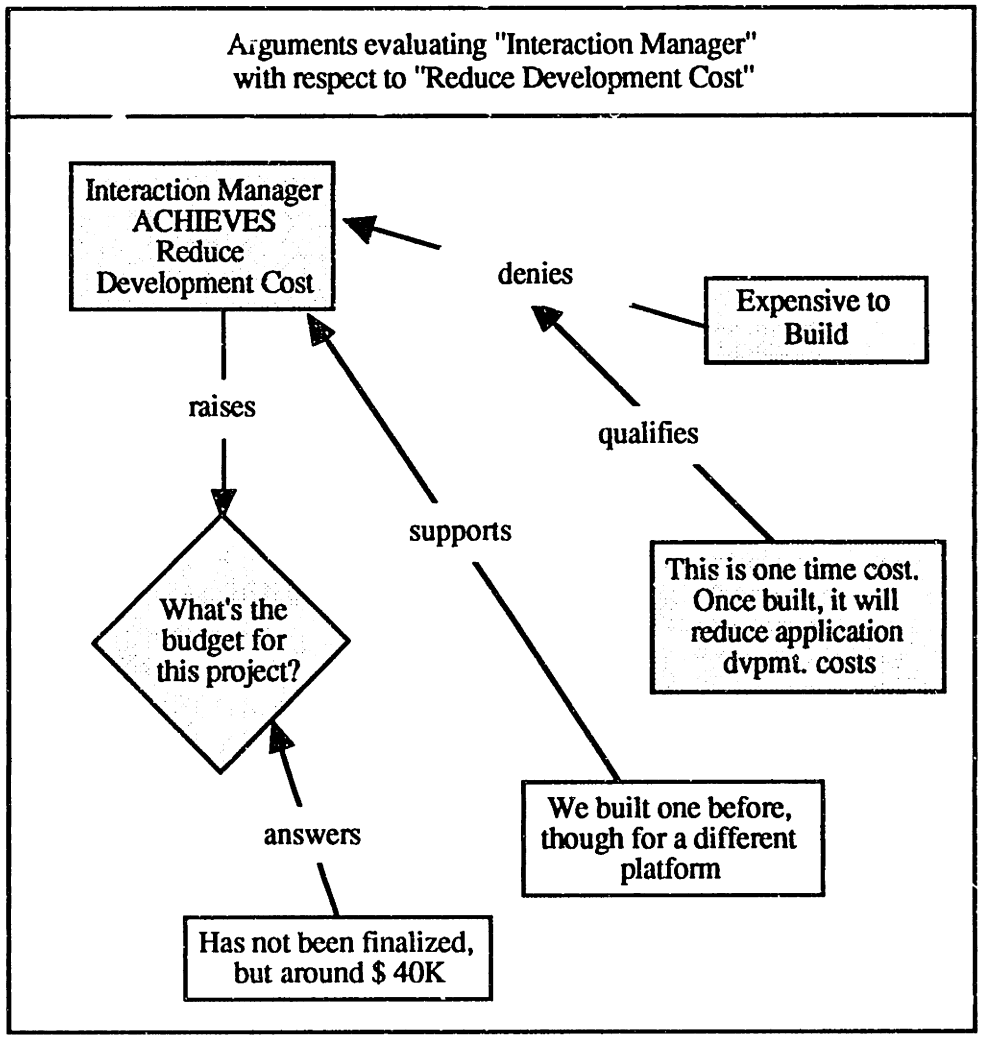


Figure 2.7 The argument browser updated with additional arguments.

The user can also relate two claims with stronger-than (or equal-to) relation if he judges one claim to be stronger than the other claim. The user would also have to specify the attribute on which they are being compared -- e.g. plausibility. In fact, the user is able to compare any two objects of the same type using these relations. The system can sometimes figure out from this comparative information which alternatives dominate others.

The consequences of additional claims or questions entered are propagated. For example, the decision matrix shown earlier would now show the relevant cells marked appropriately depending on the type of changes (e.g. marked for reevaluation).

Reusing Past Decision Rationales

User: Find me useful information from past decisions

Target System: The following decision matrix summarizes the updated evaluations, given the new knowledge transferred from past decisions. The evaluations are tentative, subject to your modifications and confirmations.[3] (Fig. 2.8)

[3] Although the system is shown as responding immediately here, the implemented system requires much interaction with the user to produce this response, as discussed in Section 7.1.2.

The Current Status of Decision Problem: What kind of application interface?				
Requirements / Alternatives	Reduce development cost	Highly portable	Common User Interface	Implement Direct Manipulation
Interaction Manager	L*#	H	M*	L*
Toolbox	M	M*	unevaluated	H*
Extensible Interaction Manager	L*	M*	unevaluated	M*

Figure 2.8 Decision matrix that displays the additional alternatives (e.g. Extensible Interaction Manager), additional goal (e.g. Common user interface), and updated evaluations as a result of transferring relevant rationales from past decisions.

User: Show me the reasons for the evaluation of Interaction Manager with respect to the criterion, Implement Direct Manipulation.

Target System: *displays the argument browser showing the arguments augmented by those that have been brought in from past decisions. Again, the shaded nodes represent those objects that have not changed since the last examination by the user.responsible for the evaluation.* *(Fig. 2.9)*

The interface is the same whether the additional objects (alternatives, goals, claims) were added by additional human or by the system through retrieving them from past decisions. The only difference is that in the latter case the changes are tentative, and they may be pruned before finally incorporating them into the current knowledge base, as described below.

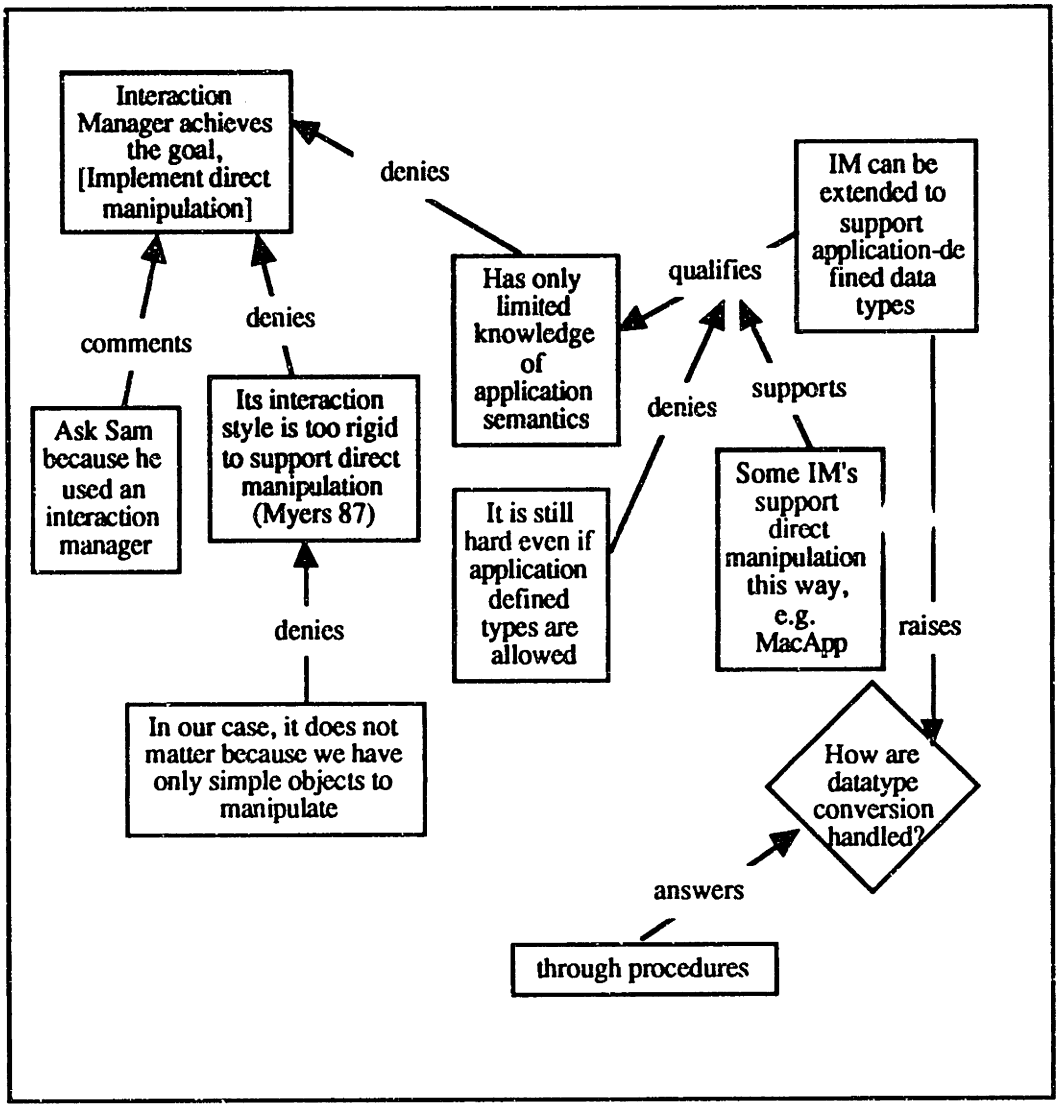


Figure 2.9 An argument browser displaying the arguments incorporated from past decisions, potentially relevant for the evaluation of the alternative, Interaction Manager, with respect to the goal, Implement direct manipulation.

The user can examine the new objects, deletes those not useful, and keeps the rest after modifying them if necessary. For example, John deletes the argument on the left middle, "Ask Sam because he used .." because the fact that Sam used an interaction manager is a context-sensitive knowledge that does not carry over to the current decision context. Also, John modifies the argument on the bottom left side, "In our case, it does not matter because we have only simple objects to manipulate," to "It may

not matter if we decide not to allow customizable icons." In other words, the original argument did not exactly carry over, but suggested something close that is relevant.

Managing Multiple Viewpoints

John wonders how evaluations would change if he gives up the goal of portability.

User: Suppose we do not consider [Portability]. How does that change the evaluation?

Target System: *shows a changed decision matrix (not shown)*

The user can examine the reasons for the changed evaluation. As before, the changed objects have been marked. Now he wants to save the current state so that he can come back to it later or compare it to other states that have also been saved.

User: Create a viewpoint, called "Without portability".

Target System: The new viewpoint will inherit from the initial viewpoint, and the changes that have been introduced since the initial viewpoint are:

- [Portability] is no longer a goal.
- The following objects have been added: [Common User Interface] (GOAL), [Extensible Interaction Manager] (ALTERNATIVE), ...
- The following objects have been inactivated: [Interaction Manager] (ALTERNATIVE), ...

- The following objects have been changed: the importance attribute of [Portability] (GOAL), ...

Is it o.k. to save this new viewpoint?

User: o.k.

Target System: The viewpoint, "Without portability," has been saved.

Wrapping Up

User: What do we need to do to resolve the current decision?

Target System:

- We need to resolve the decision, [Should the windows be supported over the net?]
- We have to answer the Question, [What are the examples of the interaction manager based systems and their experiences?]

John continues to examine the decision problem, its alternatives, arguments, in the manner described above. Finally, he asks the system to monitor a decision for him.

User: Let me know when the decision, [Should the windows be supported over the net?], is made.

Target System: o.k.

Chapter 3

The Structure of Decision Rationales

The goal of this chapter is to characterize the elements of decision rationales that are important to represent explicitly. Of course, what is important to represent depends on the task for which the representation is used. The tasks for a rationale management system have been illustrated in the scenario. In the first section, I summarize these tasks in the form of questions. Then I develop a sequence of models, each of which makes explicit the objects and the relations that make up decision rationales. The consecutive models are increasingly more complex because the objects and relations are successively differentiated from one model to the next. For each model, I discuss what extra tasks the additional refinement allows us to do. The framework provided by these models is used in the rest of the thesis, to present DRL (Chapter 4), to discuss the kinds of rationales that can be reused (Chapter 6), as well as to discuss related representations (Chapter 7).

3.1 What do we want to do with decision rationales?

One way of characterizing a task is to list the questions that we need to answer to accomplish the task. Given the goal of producing the behaviors illustrated in the scenario, our representation should be able to answer the following questions, abstracted from the scenario.

- What is the status of the current decision?
- What did we discuss last week and what do we need to do today?
- What are the alternatives being considered?
- What are their pros and cons?
- Why do we even consider this alternative, and how is it related to the one that we discussed last week?
- What are the two most favorable alternatives so far?
- How does this new fact affect the current evaluations?
- What if we do not consider this goal?
- Why is this goal important anyway?
- What are the decisions that depend on this decision?
- What are the unresolved decisions? What are we currently doing about them?
- What's the consequences of doing away with this assumption?
- How did other people deal with this problem?
- What can we learn from the past decisions?

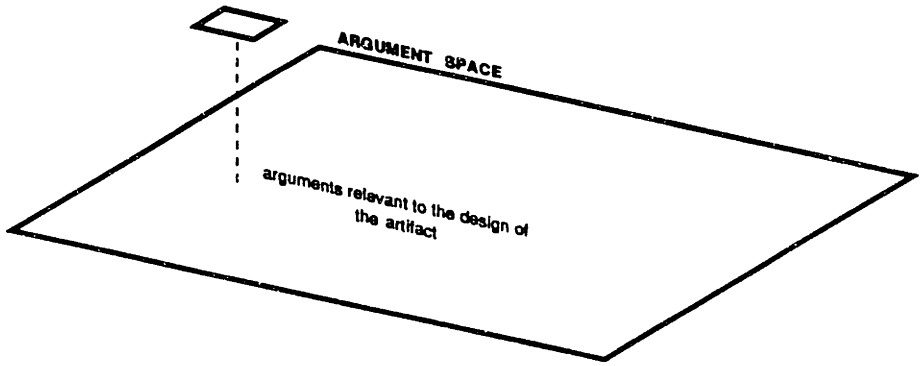
The above list of the questions is by no means complete. They have been chosen to illustrate the differences in what the following models do or do not allow us to do. They are, however, representative of the questions that arose in our experience of capturing and

managing decision rationales. As such, they serve as test cases against which the adequacy of a model can be checked. If it turns out that there are some questions which cannot be answered by a particular model of decision rationale, then answering those questions is not a task of the system based on that model or the system would have to extend its model.

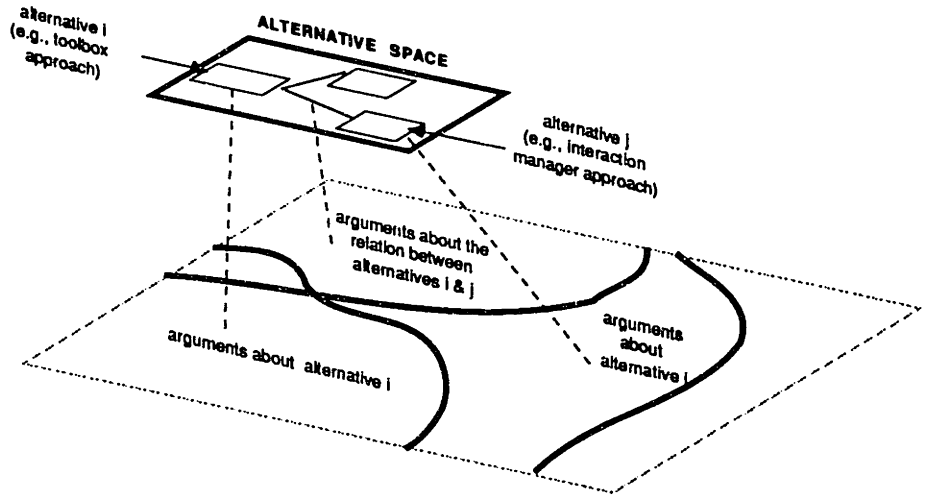
3.2 Models of Decision Rationale

We now develop a series of progressively richer models of decision rationale. As shown in Fig. 3.1, each model makes explicit the objects and the relations that make up decision rationales. The successive models are increasingly more complex because the objects in the next model in the sequence differentiate the objects in the previous one.

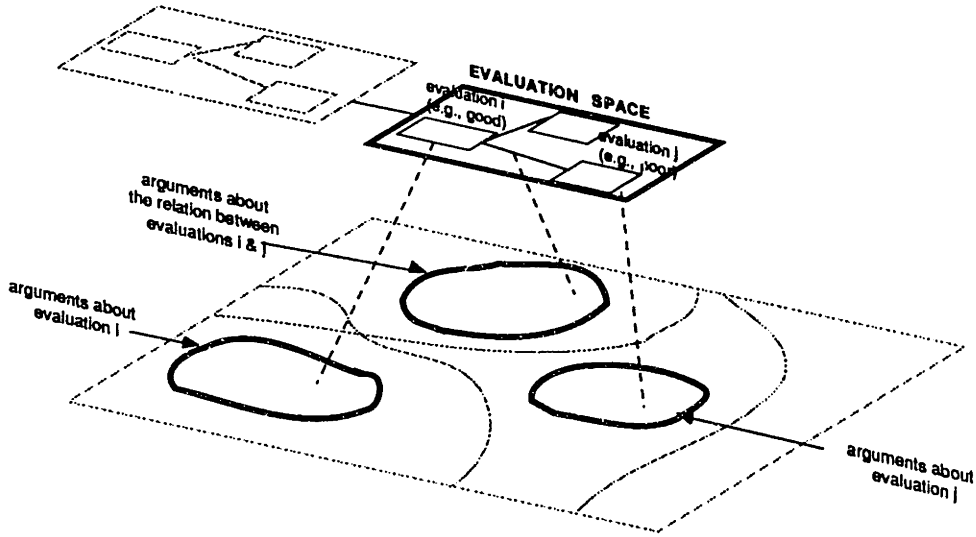
Decision rationale in the most general sense is an explanation of why the decision was made the way it was. So in our first model of decision rationale, an artifact is associated with a body of reasons as shown in Fig. 3.1a. The internal structure of these reasons can be made explicit to different degrees. At one extreme, they can be completely undifferentiated. An example is the natural language description of a historical record we used earlier. We can also imagine a representation, however, where logical support relations among reasons is made more explicit by providing constructs like *Logically Implies*, *Supports*, *Denies*, *Qualifies*, and *Presupposes*. We will use the term Argument Space to refer to what we have called a body of reasons because the reasons are captured either as a historically recorded or logically structured record of the various arguments relevant for a decision.



(a) MODEL 1: A decision outcome is associated with a body of all the arguments explaining the outcome.

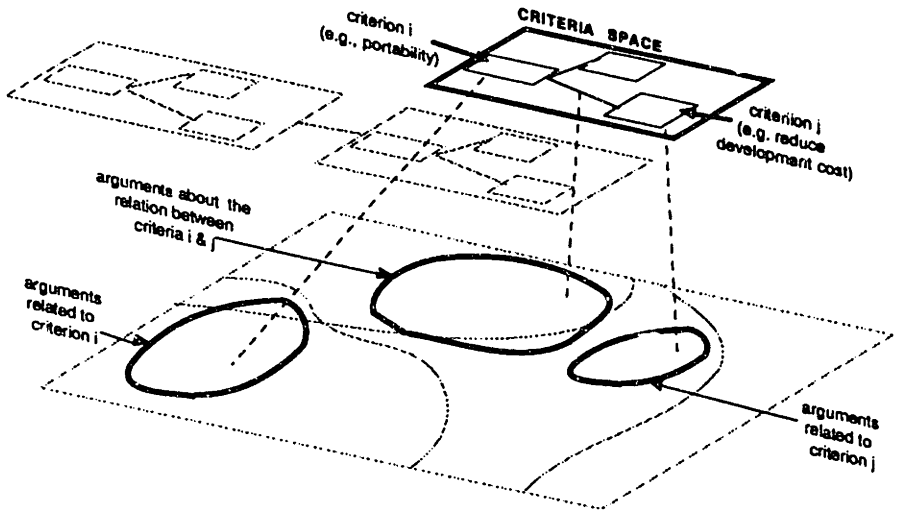


(b) MODEL 2: Alternatives and their relations are made explicit and the arguments about individual alternatives can be differentiated.

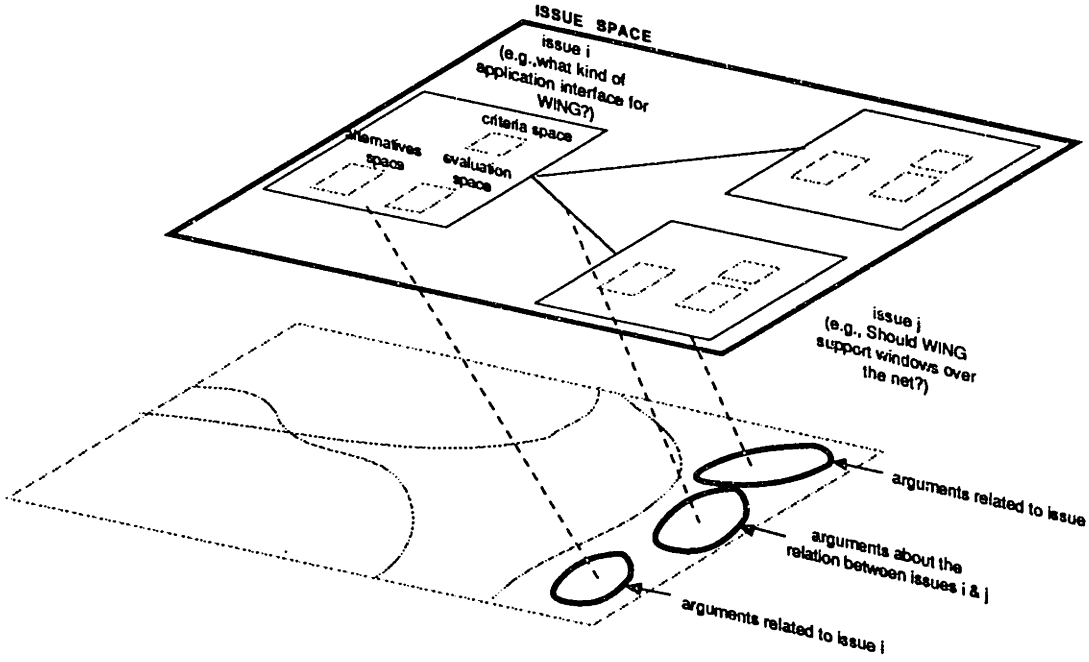


(c) MODEL 3: Evaluation measures used and their relations are made explicit and the arguments about them can be differentiated.

Figure 3.1 (a)-(c). Models of decision rationale
(continued on the next page)



(d) MODEL 4: Criteria used for evaluations and their relations are made explicit and the arguments about them can be further differentiated in the argument space.



(e) MODEL 5: Individual issues are made explicit, each of which contains the alternatives, evaluations, and criteria used in discussing the issue. A part of the argument space includes the meta-arguments about the issues and their relations.

Figure 3.1 (d)-(e). Models of decision rationale
(continued from the previous page)

There is much we can do with our first model of decision rationale. A representation based on this model can help us answer the questions, "What did we discuss last week and what do we need to do today?" Such a representation can also help us answer the questions: How did other people deal with this problem? What can we learn from the past decisions? Our first model, however, does not help us much with the other questions, though we shall qualify this statement immediately. Saying that it does not help much is not saying that we cannot answer these questions. If the user works hard enough and the representation based on the model has all necessary information captured, even if it is just in the form of natural language free text, these questions can be answered. So the real issue is how much the model itself helps us answer the questions by making the structure of the argument space more explicit for us to reason about or by providing computational services that help us answer the questions directly. We will see how more differentiated models allow us to answer these questions more easily, although they increase the cost in some other ways [Conklin & Yakemovic in print].

Our second model (Fig. 3.1b) differs from the first by making explicit multiple alternatives and their relations. Decision making involves formulating several alternatives for a problem, comparing them, and merging them as needed. In our first model, only a single solution is made explicit at a given time, and all other alternatives are present only implicitly in the argument space. Our second model makes the alternatives explicit, including the ones that have been rejected. Once the alternatives become explicit, we can talk about their attributes (e.g. current status such as "rejected" or "waiting for more information"), make the relations among the alternatives explicit (e.g. specialization, historical precedence), define computational operations on them (e.g. compare alternatives, display the alternatives that specialize another alternative), or even argue about whether an alternative is worth considering. The alternatives other than the one finally chosen are interesting because many of the issues discussed and the knowledge used in evaluating them are important in

other contexts or for re-evaluation of a discussion when situational constraints change. We use the term Alternative Space to refer to this set of multiple alternatives and their relations.

The relations among alternatives can be historical or logical. Historical relations may be not only the linear sequence that we usually describe as versions, but also more complex relations such as layers and contexts [Bobrow & Goldstein 1980]. The logical relations may include *Specializes*, *Generalizes*, *Elaborates*, or *Simplifies*. Or alternatives can be related through a design space [McKinlay et al. 1990]. To the extent that we want a representation to represent different alternatives and their relations, we say that the alternative space is within the scope of the representation.

By now, we have an alternative space connected to the argument space, as shown in Fig. 3.1b. For each of the alternatives, there are arguments describing the reasons for its current evaluation, just as in our first model there are arguments describing the evaluation status of that single alternative, i.e., that it was chosen. Some of the arguments can be shared; for example, an argument can support an alternative while denying another; so it is better to think of the arguments about the different alternatives forming a single large argument space, as shown in Fig. 3.1b.

Once the alternative space is represented, we can imagine how we can make a system that helps us answer some new questions in our list. To answer "What are the alternatives being considered?" and "What are their pros and cons?," we can associate an argument space with each of the alternatives through relations such as *Supports* or *Objects To*. If the representation of the alternative space makes explicit historical relations (e.g. *Replaces*) or structural relations (e.g. *Is A Part Of*), among the alternatives, then it can also help us

answer questions such as "Why do we even consider this alternative, and how is it related to the one that we discussed last week?"

However, once we make explicit multiple alternatives, we need to articulate more carefully what the argument space is about. In our first model when we had a single artifact, namely the chosen one, the argument space contained reasons for the choice of that artifact. Similarly, the arguments for the other alternatives are about why they were not chosen, or, to generalize, why they have their particular evaluation status, e.g. "Still in Consideration", "Waiting for More Information", "Rejected". These evaluation status could be nominal categories (such as the above examples), ordinal categories (such as "Very Good," "Good," and "Poor") or a continuous measure (such as the probability that the alternative will achieve a given set of goals).

Therefore, we introduce a new space, Evaluation Space (Fig. 3.1c), where the evaluation statuses of all alternatives are made explicit and inter-related. Usually, we do not and need not specify any elaborate relation among the evaluation measures we use. Often, the implicit ordinal relation among these values (e.g. "Very Good," "Good," "Poor," "Very Poor") is sufficient; we leave it to the human user to assign these values to the alternatives. However, if we want to define any computational service that manages these values, for example, a program that automatically propagates and merges evaluation measures to produce a higher level summary, then we need to be very careful about what these values mean. We need to specify the units of measurement, some calculus for combining them, and a model specifying what they mean. Even in the case where these actions are left to the user, for example. if the human user is expected to combine these values to produce a higher level summary measure, we need to set down what these values mean so that their interpretation does not become arbitrary.

Making the evaluation space explicit allows us to differentiate two components of the argument space: (1) arguments about why an alternative has its current evaluation status, and (2) arguments about the alternatives themselves, e.g., why we should or should not even consider an object as an alternative or whether this alternative is really a special case of another alternative. With a representation of the evaluation space, we can now answer questions such as: "What are the two most favorable alternatives so far?" and "How does this new fact affect the current evaluations?" We can also explain an evaluation measure pointing to the arguments in the argument space behind the decision in question, and elaborating on how the particular evaluation measure is derived or computed from these arguments or how it is related to other measures.

Our models so far do not make explicit the criteria used in producing an evaluation. However, the criteria used for the evaluation and their relations are usually quite important to represent explicitly. For example, it is important to know that the argument "Clean separation between window manager and application" is a pro-argument for the alternative "Interaction Manager" *because of* the goal of portability, which is used as a criterion for evaluation. By making this criterion explicit, we can group all the arguments that appeal to this criterion and weigh them against one another. If the criterion changes or becomes less important, we can perform appropriate operations on all the arguments that presuppose this goal (for example, making these arguments less important). Knowing how a criterion is related to others (e.g. knowing that "Implement direct manipulation" is a way of achieving "Naive user support"), also allows us to change its importance when related criteria change. We use the term Criteria Space to refer to these criteria and their relations. As Fig. 3.1d shows, once we have the criteria space explicit, we can further differentiate the argument space by grouping those arguments which are about the criteria and their relations.

Hence, it is important that a language whose scope includes the criteria space represent the different attributes of the criteria and the relationship among them. For example, it should allow us to represent the importance of these criteria and the synergistic or tradeoff relations among them. A set of criteria can be sub-criteria of another in the sense that satisfying them facilitates the satisfaction of the latter. These sub-criteria can be related among themselves in various ways: they can be mutually exclusive in the sense that satisfying one makes it impossible to satisfy others; or they can be independent of each other in the sense that satisfying one does not change the likelihood of satisfying others. These sub-criteria can be related to their parent criterion in various ways as well: they can be exhaustive in the sense that satisfying all of them is equivalent to satisfying the parent goal; or they may not cover the parent goal completely.

With the criteria space represented, we can now see how the system might be able to help us answer questions such as: what if we do not consider this goal? or why is this goal important anyway? The answer might be "If we give up this goal (say portability), then the evaluation of the alternative X changes to "High" because all these claims that argue against X were based on the importance of portability." or "Portability is important because it is a subgoal of another important goal, Have a wide distribution." These answers can be derived only from a representation that makes the relation between evaluations, criteria, and arguments explicit. Of course, representation of the criteria is not sufficient for answering these questions; however, the explicit representation of the criteria space seems a necessary condition if we want to provide some support answering these questions. At least, we would have the information necessary to define an operation that will give or suggest the answers to these questions. We will give examples of such operations later in Chapter 6.

So far, we have identified and discussed the structure of a *single* decision underlying an artifact, namely which of the alternatives should we choose? However, with the

representation of such local structures alone, namely its argument space, alternative space, and criteria space, we still cannot ask some of the questions in the list such as: "What are the unresolved issues?" and "What are the issues that depend on this issue?" To answer such questions, we need a more global picture of how individual issues are related. A decision often requires and/or influences many other decisions. For example, a decision can be a sub-decision of another if the latter requires making the first decision. A decision can be a specialization of another if the first decision is a more detailed case of the second. It is important to capture how these decisions are related, and we use the term Issue Space to refer to them.[4] A unit in this issue space is, therefore, a single decision that has as its internal structure the other spaces, as shown in Fig. 3.1e. Once we have an issue as an explicit element, we can associate attributes such as "Status" and "Actions Taken" with issues and answer questions such as "What are the unresolved issues?" as well as "What are we currently doing about them?" Representing the dependency relation among the issues will allow us to answer questions like "What are the issues that depend this issue?"

There are still some questions that we have not yet covered such as: How did other people deal with this problem?" and "What can we learn from the past decisions?" We argue, however, that the five spaces so far identified, the spaces of arguments, alternatives, evaluations, criteria, and issue, can contain enough information to answer these questions. In Chapter 6, we discuss computational operations that help us answer these additional questions by exploiting the structure of the five spaces.

[4] The term Issue Space is used instead of Decision Space because Decision Space is sometimes used to refer to the set of all objects relevant to making a decision.

Chapter 4

DRL: A Decision Representation Language

This chapter presents a language, DRL, developed for representing and managing decision rationales. DRL has been designed based on the last model in the sequence of the models developed in the previous chapter. As such, it provides constructs for representing the five spaces of argument, alternative, evaluation, criteria, and issue.

The philosophy underlying the design of DRL has been a minimalist. For each of the spaces, DRL started out with the fundamental object type and the relation types that are important to answer the kinds of questions enumerated in the previous chapter. For example, for the criteria space, DRL provides the object type, GOAL, to represent a criteria and the relation, IS A SUBGOAL OF, which is necessary to answer the question, Why is portability important? In the course of using DRL to represent decision rationales,

however, additional questions arose. And it is only when these questions could not be answered by the existing constructs that additional constructs were added. Figure 4.1 shows all the constructs that DRL currently provides. Although not large in number, these constructs have proved to be adequate in representing the cases and providing the services that we have explored so far.

I first give an overall description of DRL and then discuss, in the next section, how these constructs map to the five spaces. More detailed descriptions of the DRL constructs and their semantics are provided in Appendix I. I would like to emphasize that this chapter describes DRL as a language, not the user interface for using the language. At times, the language may seem complex, but much of these complexities can be hidden from the user through user interface such as context sensitive menus that display appropriate actions in appropriate contexts. The actual user interface is described in the next chapter.

4.1 Overview

This section provides a high-level overview of DRL, and the next section discusses the constructs DRL provides for representing each of the spaces discussed in the last chapter. Figure 4.1 shows the object types that form the vocabulary of DRL. Objects of type *DRL Relation* and its subtypes can be used to link two other objects. For instance, *Achieve* objects can be used to link an *Alternative* object to a *Goal* object. The legal types that can be linked are shown inside the parentheses following the names of the relations. Figure 4.2 shows the schema for a decision graph, which is used to graphically illustrate the rationales at a given moment. This schema shows how the DRL constructs are related to one another. Figure 4.3 shows a decision graph for an example decision problem.

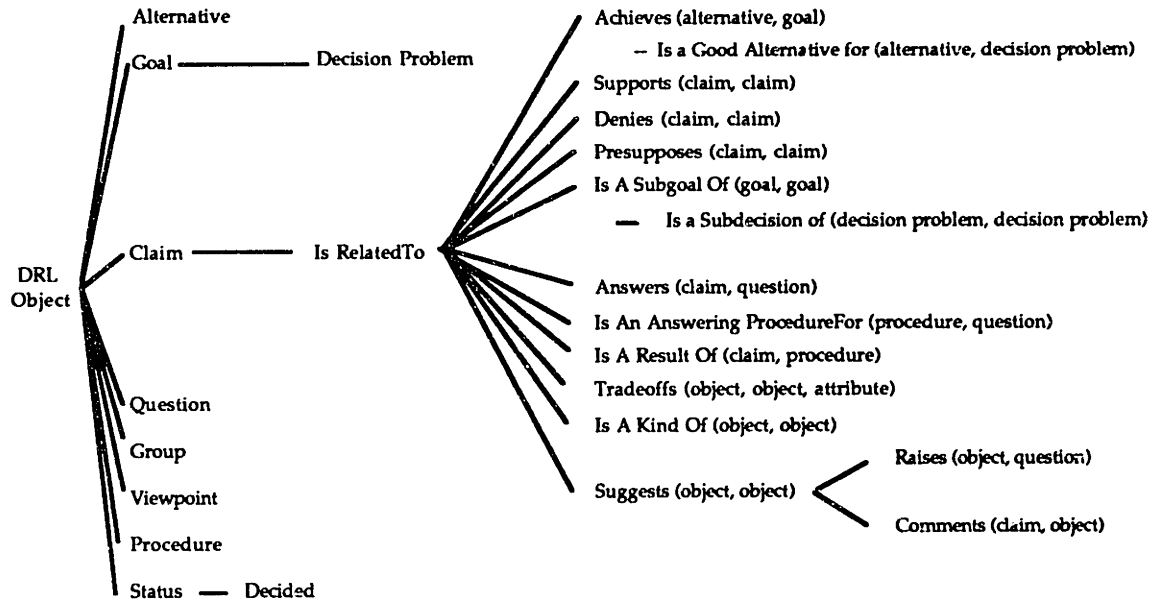


Figure 4.1 The DRL Vocabulary

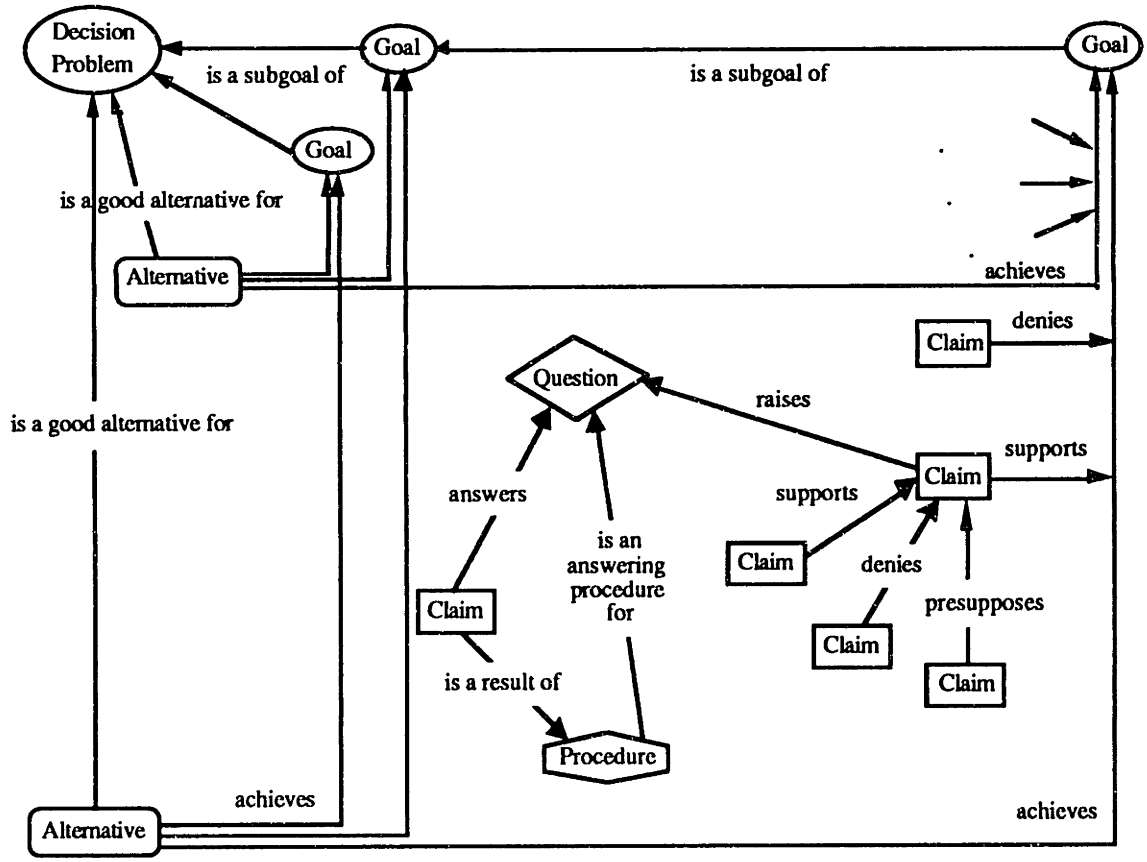


Figure 4.2 The schema for a decision graph, showing the relationship among the major constructs of DRL

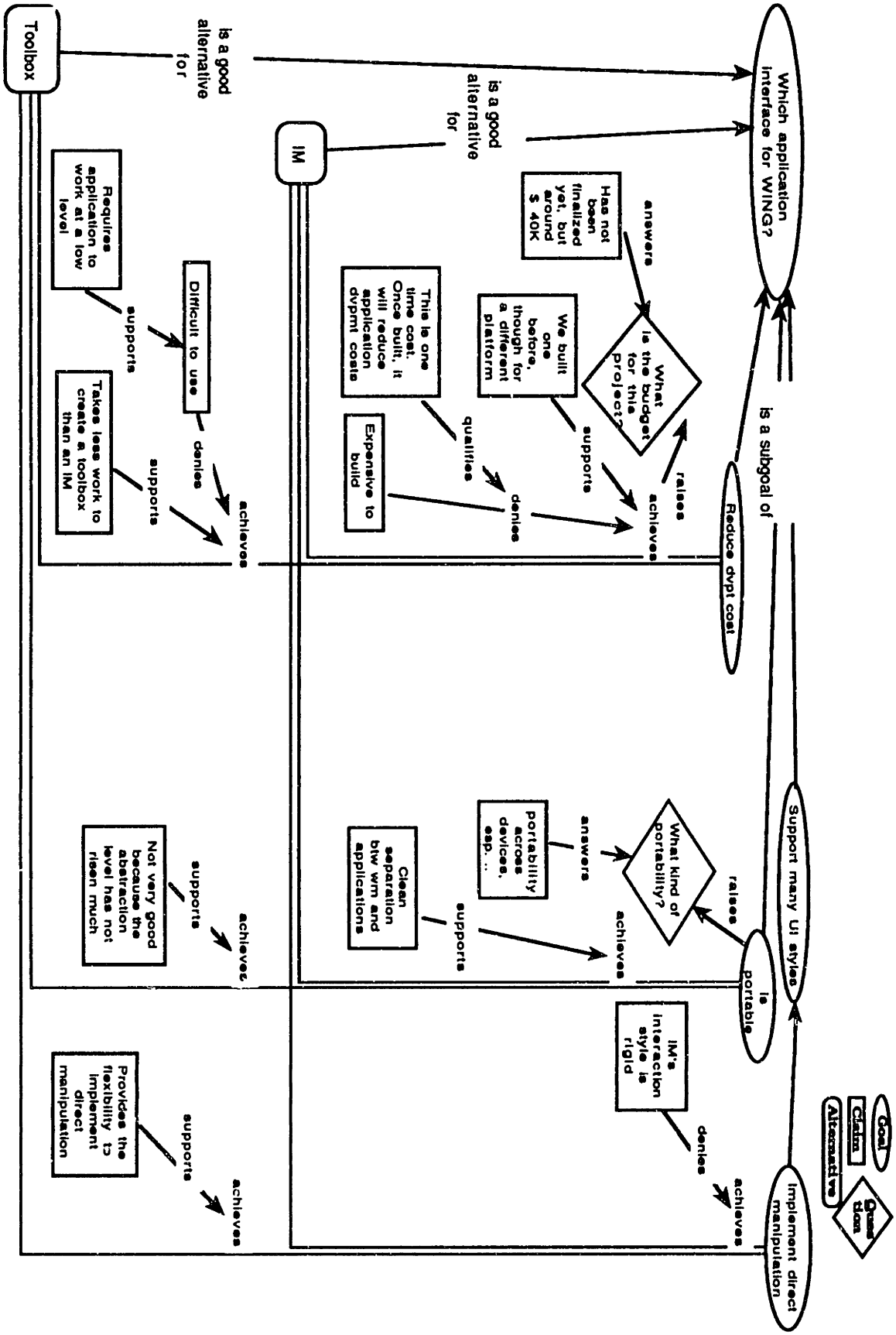


Figure 4.3 An Example Decision Graph in DRL

A *Decision Problem* represents the problem that requires a decision; for example, where to place the window commands. An *Alternative* represents an option being considered: e.g. [IM] or [Toolbox]. A *Goal* represents a desirable state or property used for comparing the alternatives. A *Goal* is elaborated in terms of its subgoals. In particular, a decision problem is a goal of special kind, i.e. of the form, "Choose the optimal X for Y," where Y is elaborated by its subgoals. For example, choosing the best alternative for the decision problem, [Which application interface for WING?], means choosing an alternative that satisfies as much as possible its subgoals: [Reduce dvpt cost], [Is portable], [Support many UI styles]. Every relation in DRL is a subclass of *Claim*, as shown in Fig. 4.1. For example, the rightmost *Achieves* link in Fig. 4.3 represents the *Claim* that the *Alternative* [Toolbox] achieves the *Goal* [Implement direct manipulation].

We evaluate an *Alternative* with respect to a *Goal* by arguing about the *Achieves* relation between the *Alternative* and the *Goal*, i.e., the claim that the *Alternative* achieves the *Goal*. We argue about a *Claim* by producing other *Claims* that *Support* or *Deny* the *Claim* or by qualifying the *Claim* by pointing out the *Claims* that it *Presupposes*. The overall evaluation of an alternative is represented by the Evaluation attribute value of the *Is A Good Alternative for* link between the *Alternative* and the *Decision Problem*, i.e., the claim that the alternative is a good alternative resolution for the issue. This evaluation is a function of the evaluations of the *Achieves* claims that link the *Alternative* to the different *Goals*.

The constructs of DRL can be divided into two major categories: those specific for each of the five spaces and those that are generic. The constructs discussed so far are specific to a space. DRL also provides constructs for representing objects and relations that are useful in any of the spaces. *Group* is used to represent a collection of objects and has the attribute, "Member Relations," which tells us how the objects are related. A relation can take a *Group* of objects rather than a single object. For example, a *Goal* may be related to a *Group* of

other *Goals* through a *Is A Subgoal of* link. The construct, *Is A Kind Of,* represents the usual generalization (or specialization) relation. The other constructs are discussed in the next section and in Appendix I.

4.2. DRL's Representation of the Decision Rationale Model

This section discusses the DRL constructs in more detail and explain how they represent each of the five spaces in the model that DRL adopts, namely the argument, the alternative, the criteria, the evaluation, and the issue space. Figure 4.4 shows the regions of a decision graph schema which represent the different spaces.

The Argument Space

An argument is represented in DRL as a set of related *Claims*. A *Claim* subsumes what other people might call facts, assumptions, statements, or rules. Instead of making these distinctions, which is sometimes arbitrary and difficult to make, a DRL *Claim* has the attribute Plausibility that indicates how much confidence we have in the claim. This has the advantage of not imposing a set of predetermined categories on the user, and avoiding the ambiguity resulting from the disagreement among people on what facts or assumptions are. When it is desirable to make the distinction say, between facts and assumptions, we can do so simply by specializing a claim or by using nominal categories like "fact" and "assumption" based on the Plausibility attribute values in different *Claims*. In fact, we can do so post facto or dynamically by using a numeric measure as the plausibility value, and mapping between this measure and the measure based on nominal categories like "facts" or "assumptions."

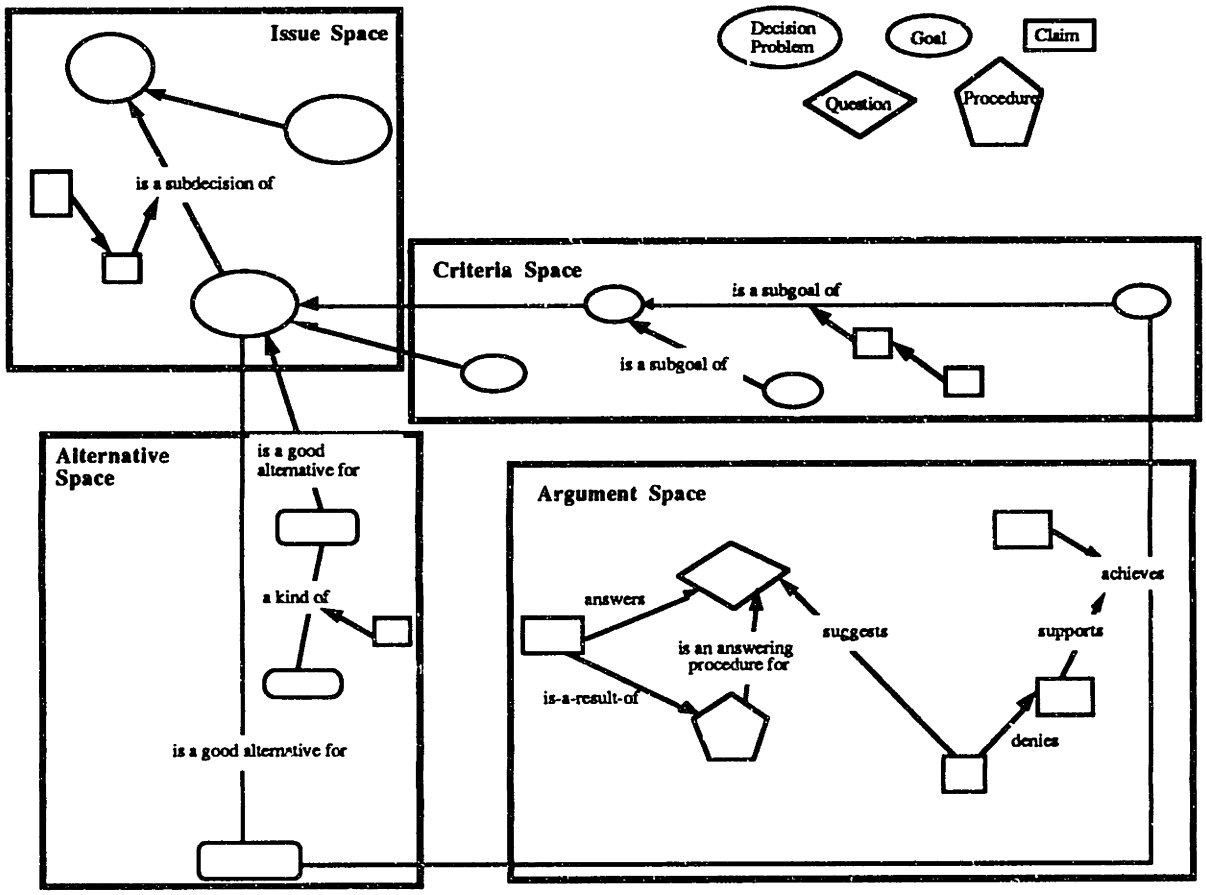


Figure 4.4 The constructs of DRL grouped by the kinds of decision rationales that they represent.

A *Claim* can be *Supported, Denied,* or *Presupposed* by another *Claim.* For example, the *Claim*, [Difficult to use], is supported by the claim, [Requires application to work at a lower level]. All DRL relations are special types of *Claims.* For example, when we link *Claim* 1 to *Claim* 2 through a *Supports* relation, we are making the claim, [Claim 1 *supports* Claim 2]. Likewise, an *Achieves* relation from an *Alternative* object to a *Goal* object represents the claim that the alternative achieves the goal. Hence, any DRL relation, like *Supports, Denies, Achieves, Is A Subgoal Of,* is a *Claim*, and can be argued about; users can support, deny, or qualify them. For example, [Interaction manager achieves Reduce dvt cost] (shown as the achieves link between [IM] and the goal [Reduce dvpt cost]) is a claim which is denied by [Expensive to build]. That [Expensive to build] *denies*

the achieves claim is itself a claim, which is in turn denied by the claim [This is one time cost].

The Alternative Space

Alternative is the fundamental unit of the Alternative Space and represents an option being considered, e.g. "Toolbox" and "Interaction Manager." Alternatives are shown as rounded boxes in the left bottom of the figure. Currently, in DRL, alternatives are related to each other through only the generic relation, *Is A Kind Of*. Thus, we can say that [Extensible IM] is a special case of the alternative, [IM]. There may be other relations that are important to represent among alternatives, such as *Elaborates*, *Simplifies*, or *Is the Next Version of*. Although these relations may be added to DRL if the task requires them, they are not in the core vocabulary of DRL.

DRL represents the arguments about the alternative space the same way it represents the arguments about the goal space. We can argue about whether an alternative should be an alternative at all or whether an alternative is really a specialized version of another alternative; we can do so by creating *Claims* that deny or support the appropriate relations, such as *Is A Good Alternative for* or *Is A Kind of*. The relations can be also qualified by linking them to another claim via a *Presupposes* relation. For example, the user may want to argue that [Extensible IM] is really not a special kind of [IM] because the extensibility violates the fundamental characteristic of [IM], namely the strict separation of window manager and application. He can express this argument by adding a claim to that effect and making it deny the *Is A Kind Of* relation between the two alternatives.

The Criteria Space

In DRL, criteria are represented by *Goals*. DRL uses the term "Goal" rather than "Criterion" because for each criterion, we can always define a corresponding goal, namely the goal of achieving the criterion. We rather want to convey the richer relationship possible among these goals than what the term criteria usually conveys. For example, a *Goal Is A Subgoal of* another *Goal* if achieving the first *Goal* facilitates the achievement of the second. A set of subgoals can be related among themselves in various ways; they can be mutually exclusive, independent of each other, or partially overlapping. These relationships are represented by creating a *Group* object and specifying these *Goals* to be its members; the relations among these *Goals* are specified in the "Member Relations" property of the *Group*.

Decision Problem represents the goal of choosing the best alternative. All the other goals for a decision are subgoals of the decision problem in the sense that they elaborate what it means to choose the best alternative. In our example, the interpretation of the decision problem [Which application interface for WING?] is the desired state of having chosen an application interface that is best for our window manager, and the interpretation of the goal, [Is portable], is the desired state of having chosen an application interface that makes our window manager portable. Hence, [Is portable] is a subgoal of the decision problem in this interpretation.

The arguments about the criteria space are represented in the same way as those about the alternative space. Because the *Is A Subgoal of* relation is a *Claim*, as is any other DRL relation, we can argue about whether a goal is desirable or whether it contributes to achieving another goal by arguing about this relational claim. For example, we can argue about whether [Support many UI styles] should be a goal at all by producing claims

supporting or denying the *Is A Subgoal Of* relation between the goal and the decision problem.

The Evaluation Space

The Evaluation of an object is represented in DRL as the Evaluation attribute value of the object. In particular, the evaluation of an alternative with respect to a given goal is represented by the Evaluation attribute value of the Achieves relation between the alternative and the goal. The overall evaluation of an alternative is represented by the Evaluation attribute value of the *Is A Good Alternative For* relation between the alternative and the decision problem. This value represents the evaluation of how well the alternative satisfies the overall goal. This value, in turn, is a function of the evaluations of the *Achieves* relations between the alternative and the subgoals of the decision problem. It is also a function of how the subgoals interact to satisfy the parent goal, such as the extent to which tradeoffs and synergies exist among these goals.

As default evaluation measures, DRL provides simple ordinal categories: H (High), M (Medium), and L (Low). However, the user can adopt other nominal, ordinal, or numeric measures. If the user wants these evaluation measures to be automatically computed, for example, to produce the evaluation of *Is A Good Alternative For* from the evaluations of the *Achieves* relations, then the user needs to specify the method for propagating and merging evaluations. DRL, as a language, is not committed to any particular method for doing so, but only provides an interface for using such a method. A claim has the attribute Evaluation Method which can be used to specify a method for computing the evaluation of that claim. This method can be specified globally for all the instances of Claim, but can be overridden by a specific method for any instance of Claim.

The arguments about the evaluations of claims are captured by claims as well. For example, the arguments about the overall evaluation of an alternative is represented as a set of claims that support, deny, or qualify the *Is A Good Alternative For* relation. The arguments about the evaluation of an alternative with respect to a goal is captured as a set of claims about the *Achieves* relation between the alternative and the goal.

The Issue Space

Decision Problem is the unit of the issue space. A decision problem *Is A Subdecision of* another decision problem if resolving the latter requires resolving the first. For example, deciding which application interface is best for a window manager might require deciding whether the window manager should support windows over network. A decision problem *Is A Kind Of* another decision problem if the first decision problem is a special case of the second: e.g., deciding which application interface is best for a window manager is a special case of deciding the external interface for a window manager. Again, there are other relations among decision problems that might be potentially useful to represent. For example, the *Replaces* or *Elaborates* relations might be important for describing how decisions have been elaborated over time. As yet, however, the tasks to which DRL has been put did not require them. Arguments about the issue space are represented, as usual, by the claims about the relations about decision problems, namely the *Is A Subdecision Of* and *Is A Kind Of* relations.

Chapter 5

SIBYL: An Environment for Using DRL

As described in Chapter 1, a rationale management system consists of three components:

- a language for representing the important elements of rationales
- a method for using the language to capture the rationales
- a set of services that use the captured rationales to support decision making

The first component, language, was described in the previous chapter. In this chapter, we discuss the second component of our rationale management system, namely the method for using DRL to capture rationales. The third component, a set of decision support services, is described in the next.

5.1 Overview

A method of using a language for representing rationales should tell us the following:

- when the language should be used at all, i.e. the scope of the applicability of the language.
- for a process within this scope, a description of its major phases so that the user of the language knows which constructs to use and how to use them in each of these phases. In other words, the method should tell the user how rationales (such as shown in Fig. 5.1) can be constructed from the beginning to the end so that these rationales can be examined by human users and used by the computational services.

The answer to the first question, i.e. the scope of the applicability of DRL, is that DRL is most useful in asynchronous distributed decision making, where the members of a decision making group communicate either via a shared knowledge base or email. The use of DRL, in fact the use of most structured representations with typed objects and relations, require a fair amount of cognitive processing on the part of the user. In order to express something in a typed language, one has to do at least three things: first one has to find the type in which to express it, one has to relate it to the existing types, and one has to actually encode it into the format required by the system (e.g. create a node and key in some of its attributes).

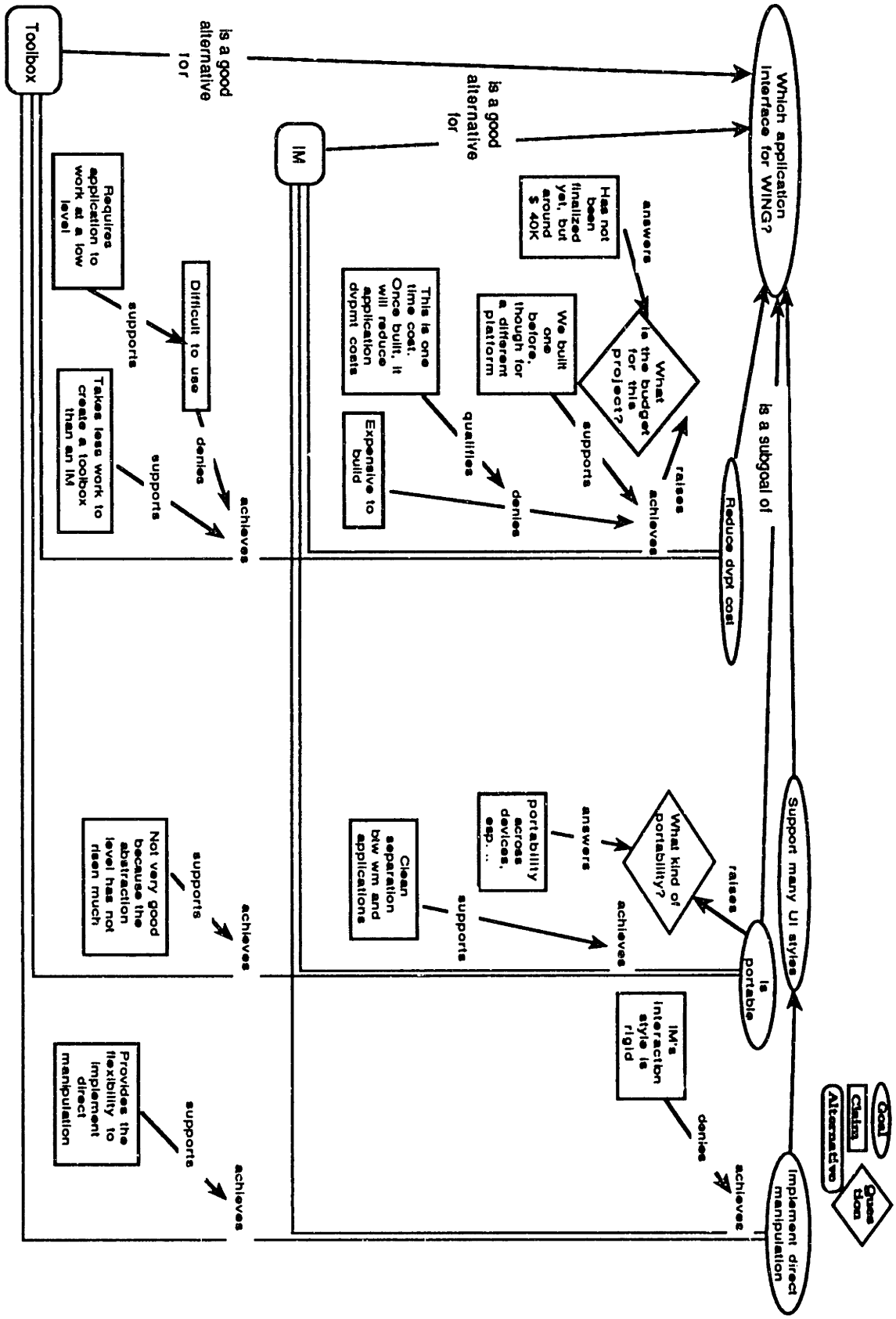


Figure 5.1 An example decision graph, reproduced from Fig. 4.3

The more such cognitive processing is required, the better it is done in the context of an asynchronous distributed decision making. In that context, users have more time to reflect on what they are expressing and on how it relates to the other existing objects. This especially valuable if users either cannot get together physically or find it less convenient to do so. DRL, I believe, is structured enough to require a fair amount of cognitive processing. Therefore, I consider its ideal context of use to be that of distributed decision making. For example, judging from the attempts to encode the rationales that unfold in real time face-to-face meetings, the kind of cognitive processing mentioned above would either not keep up with the fast pace that typically characterize such meetings or slow down the pace to a degree that might not be acceptable to the members of a decision making group. Whether DRL can be used for dynamic, real-time rationale management is an empirical question, and until it is proven otherwise, the scope of the applicability of DRL is restricted to that of asynchronous distributed decision making.[5]

The method for using a rationale representation should also provide us with a process description, from beginning to end, of the major phases involved in the rationale capture process within its scope. By "major phases" I do not mean steps of decision making process in general, but the stages that a given method introduces in supporting the decision process. That is, the phases into which a decision process is divided depends on whether the method requires different treatments for them, such as use of the different constructs and different activities. If, on the other hand, our method is the paper and pencil method that says "write down everything that is said in the meeting," discussed earlier as a simple case of an RMS, then as far as this method is concerned, there is only one major phase,

---

[5] Decision making, however, even asynchronous ones, often involves face-to-face meetings, and a rationale management system should be able to represent in some way the rationales in such meetings lest it leaves a gap within the rationales it is capturing. In SIBYL, it is suggested that the rationales that cannot be captured in the process be reconstructed after recording the process in an unstructured form, e.g. in video or audio recording.

namely the decision making process as a whole. On the other hand, if a method tells us to write down only those things that are discussed in some parts of the process, e.g. discussion sessions, then those parts would be a separate phase. Making explicit method-specific phases makes clear the process model that the method is assuming or imposing; this helps evaluate the method and relate it to other methods.

Figure 5.2 shows the major phases that SIBYL imposes on using DRL. Circles represent major states marking the boundaries of the phases:

- initialized decision structure,
- released decision structure,
- augmented decision structure,
- decided,
- decision outcome evaluated.

Between these states are the actions that bring about these states, represented as boxes in the figure. The precedence relation among these actions is indicated with a directed arc. That is, if an action requires another action to have been performed in order for it to be executable, there is a directed arc from the second to the first. Otherwise, the actions can be executed in any order and multiple times. In the following, each of these phases is described.

Before I do so, a note on implementation is in order (because a large part of the method of using DRL is built into its user interface). A method for using a language can be specified in different ways. One way is to embed the method in the interface for using the language so that the user only need to follow what seems obvious at each point. For example, the system may be able to display a context-sensitive menu that displays a possible set of actions that the user can perform at every step. These actions may embody legal ways in

which the language can be used by the user in that context. SIBYL tries to do that most of the time. Therefore, for each of the major phases, the actual interface of SIBYL is described to show how the method for that phase is being encouraged or enforced by the interface.

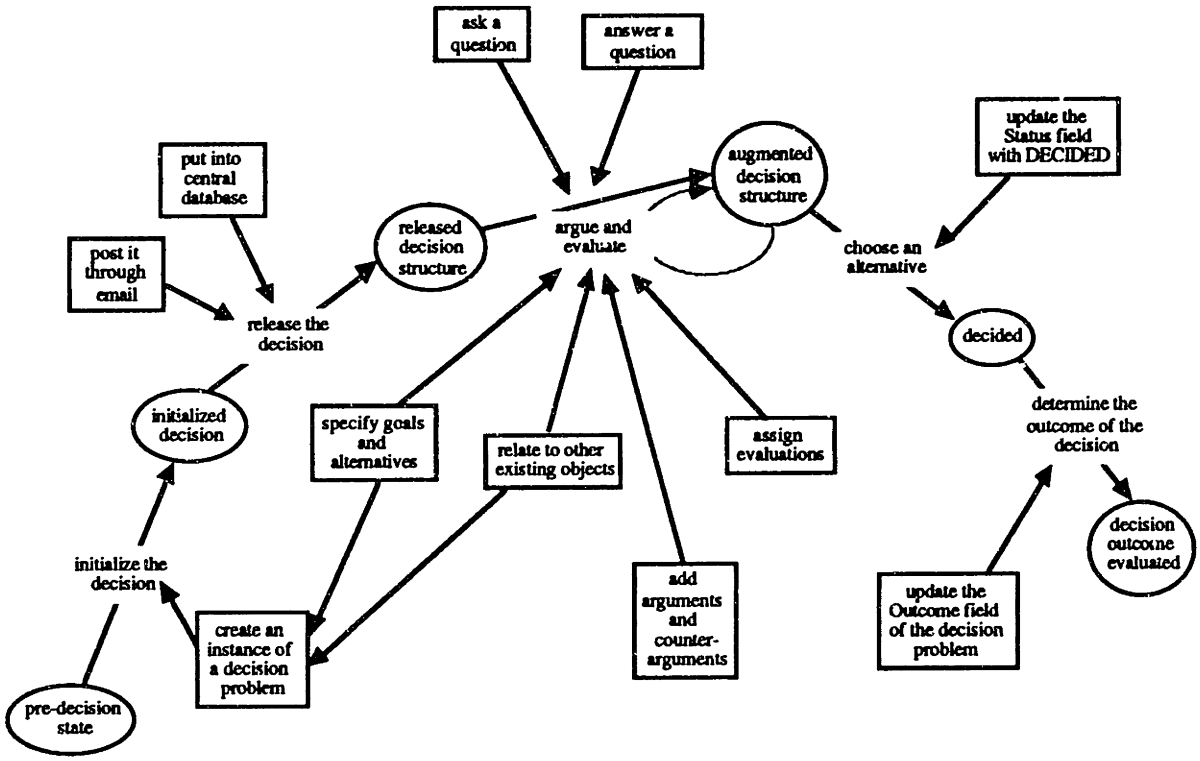


Figure 5.2 The phases in the decision making process using SIBYL

5.2 Implementation

SIBYL runs on top of Object Lens [Lai et al. 1988], a generic tool for building computer-supported cooperative work applications. That is, all the DRL types like goal and alternative have been specialized from the system object of Object Lens, THING. SIBYL also makes use of many features of Object Lens, like Rules, Agents, and different display

formats to provide its services, as described below. Conversely, all the features of Object Lens are available in SIBYL. Versions of Object Lens have been implemented in MacIntosh Common Lisp and, earlier, in Interlisp.

There are several implementations of SIBYL. All but one runs on the MacIntosh computer. One version runs in Interlisp on Xerox 1109 Lisp Machines and runs on the Interlisp version of Object Lens. There are several versions running on MacIntoshes, which are compatible with different versions of Object Lens. In the following description, I describe primarily the most current version except for the features implemented only in older versions. SIBYL has been used for capturing two real-life decision rationales and about eight cases of rationales reconstructed from case studies.

5.3 Setting up an Initial Structure

To initiate a new decision in SIBYL, one performs the following actions:

- create an instance of decision problem,
- relate the decision problem to other existing objects, such as other decisions
- specify an initial set of goals and alternatives,
- optionally, add claims and counter-claims evaluating the alternatives with respect to the specified goals

Creating an Instance of Decision Problem

In Object Lens, one creates an instance of a given type by double mouse-clicking on the type object in the type browser. The type browser is shown in Fig. 5.3 and displays all the

SIBYL types that currently exist. Double-mouse clicking on a type brings up a a template editor displaying the new instance as well as a set of menu items representing the actions that can be legally performed on the object . Figure 5.4 shows such an editor that contains an instance of *Decision Problem.* Initially, this new instance has only those attributes filled in for which default values could be obtained.

Relating the Decision to Existing Objects

The decision problem is then related to other existing objects. For instance, our example decision, "Which application interface for WING?", is a subdecision of the decision, "How best design our window manager, WING". To express this relationship, one mouse-clicks on the appropriate field (i.e. is a subdecision of) and chooses the action Add Link from a pop-up menu that appears. The action Add Link is a means of embedding a link, i.e. a

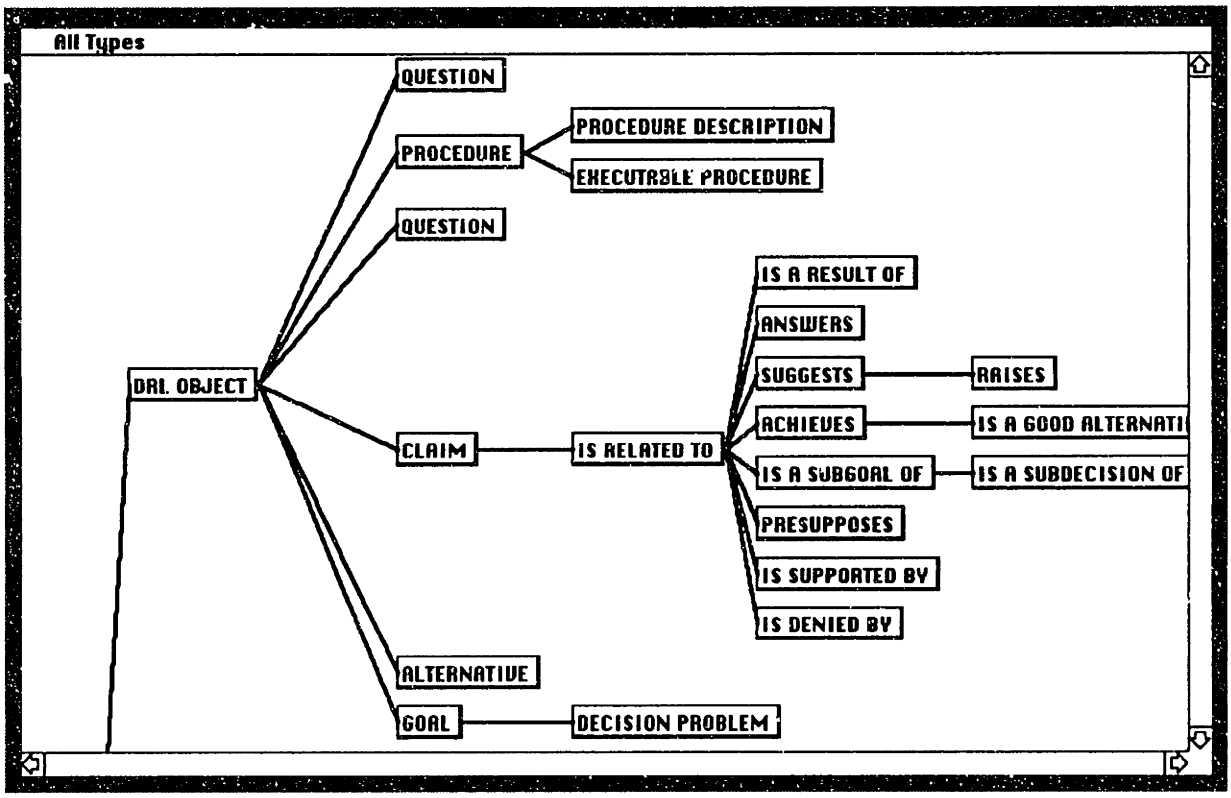


Figure 5.3 A type browser showing DRL objects

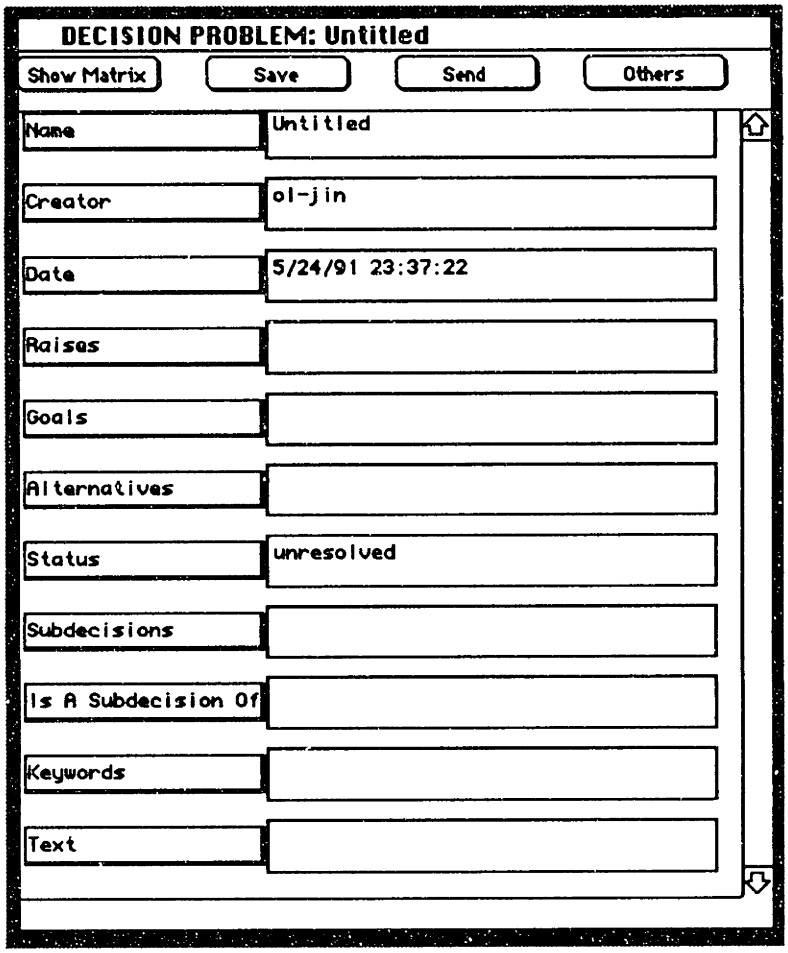


Figure 5.4 A new instance of decision problem with default values.

reference pointer, to another object within a field of an object. When the action Add-Link is selected, the cursor turns into a cross-bar and the user is asked to click on the object that should be the value of the field originally clicked on (i.e. *is a subdecision of* in our example). When the user clicks on the parent decision, a link to this decision, represented by its name in a square bracket, is embedded in that field, as shown in Fig. 5.5. A corresponding effect takes place in the parent decision problem object; its subdecision field now contains a link to "Which application interface for WING?" In Object Lens, a link to an object is shown as its name enclosed in a square bracket, e.g. [How best design our window manager, WING?]. When double clicking on this link, the object that it links to is displayed in full, as indicated by the arrow in Fig. 5.5.

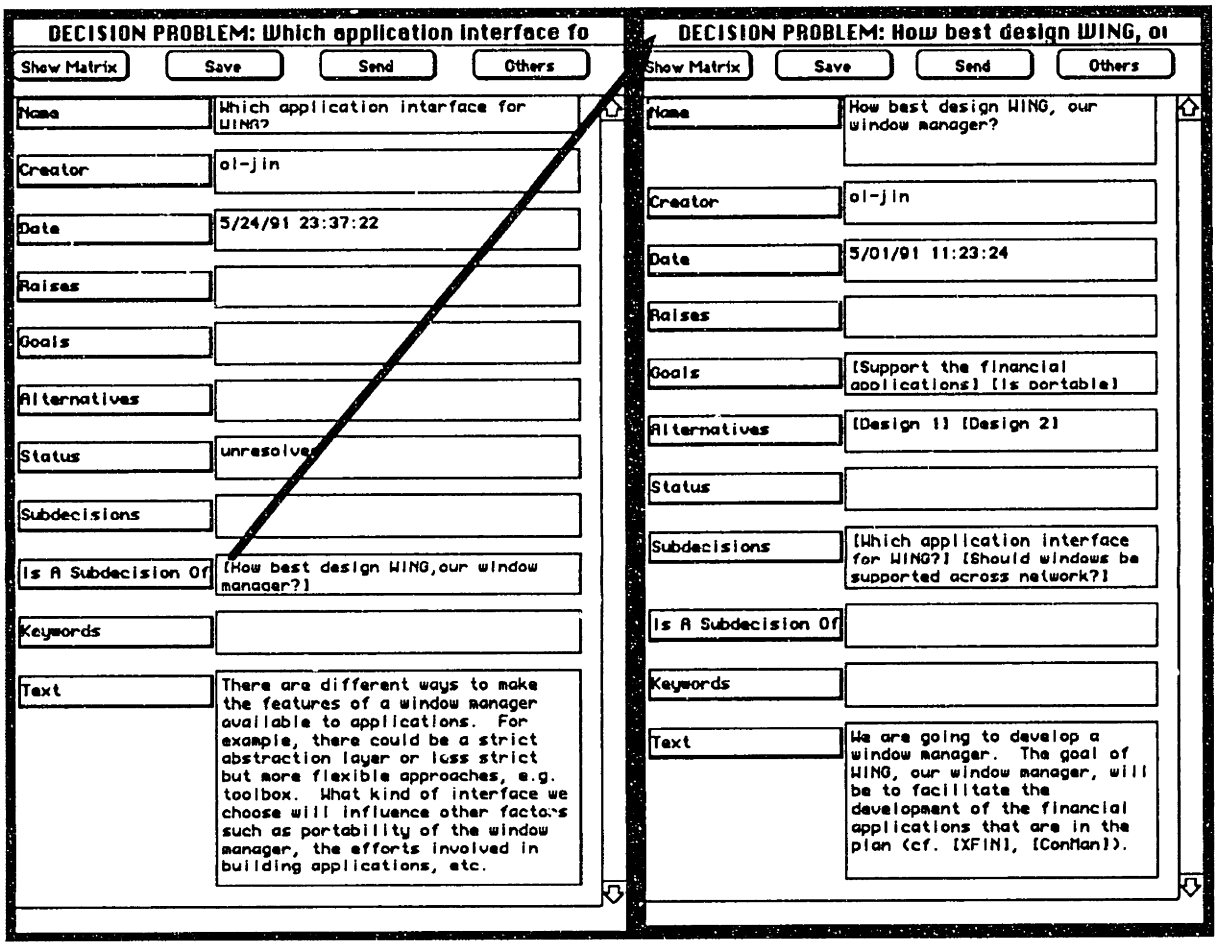


Figure 5.5 The decision problem instance related to other existing objects. In particular, it is specified to be a subdecision of [How best design WING, our window manager?]

Specifying Initial Alternatives and Goalsl

To specify alternatives for this decision problem, the user chooses the field Alternative and then mouse-clicks on the action, Add An Alternative. This action will automatically bring up an instance of a new alternative template. As shown in Fig. 5.6, this action places a link back to the decision problem in the Decision Problem field of the alternative, and a corresponding link to the alternative being created in the Alternative field of the decision problem. Here, two alternatives, [IM] and [Toolbox], have been added this way. Similarly, through the action, Add A Goal, one creates and interconnects Goal objects

standing for the properties that an optimal alternative should have. Figure 5.6 shows two goals, "Reduce dvpt Cost" and "Is portable" that have been added. [6]

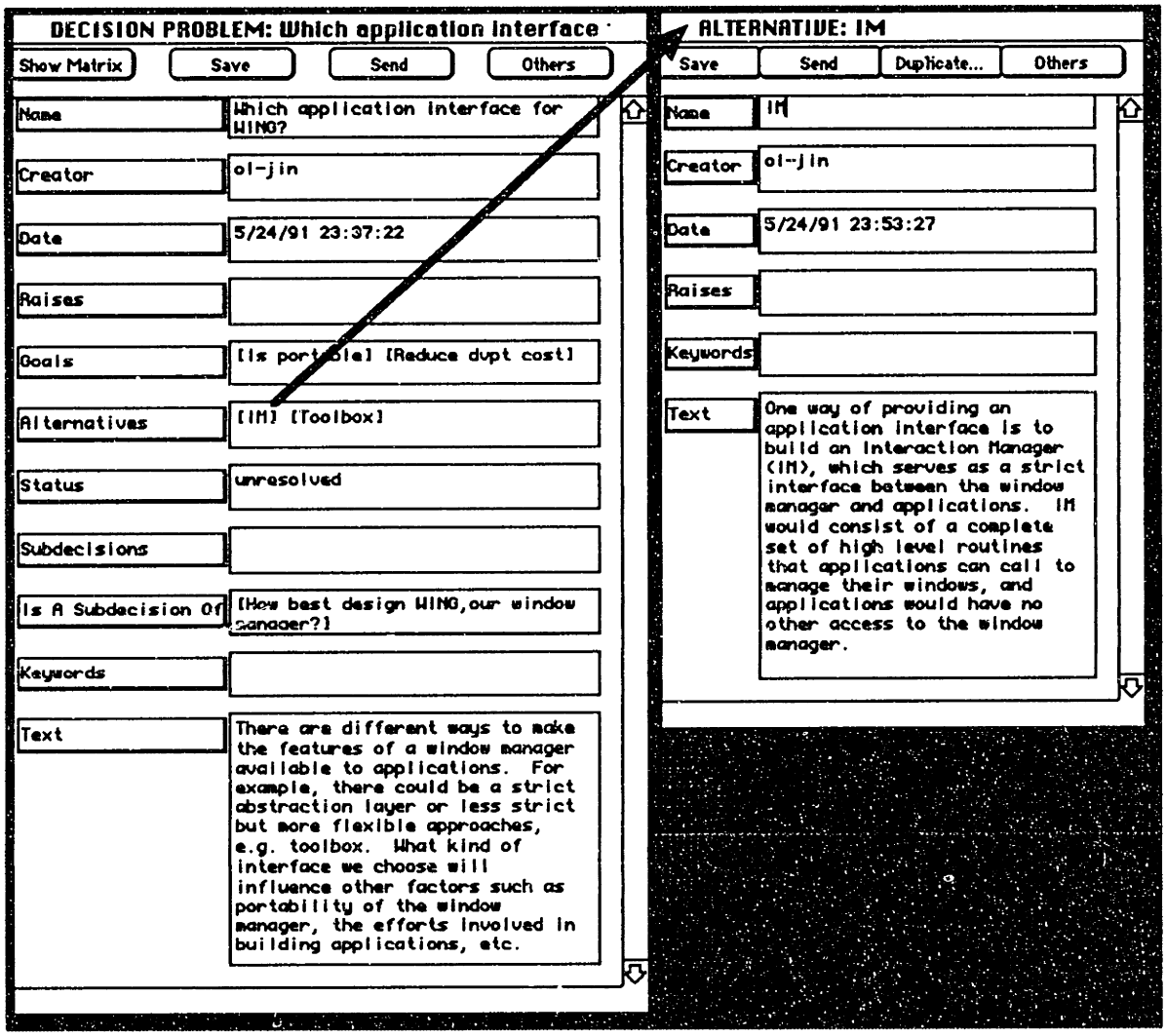


Figure 5.6 The decision problem elaborated further with its alternatives and goals. One of the alternative, [IM], is shown expanded.

The user who created the decision problem can at this point optionally elaborate on the initial set of goals and alternatives, or add arguments and counter arguments about them.

---

[6] Add a Goal, Add an Alternative actions are from the older Xerox version of SIBYL. In the current version, Add-Link is used to be consistent with the Object Lens interface. We describe the older interface here because it is simpler to explain, and for the same reason, the current version plans to implement these actions.

However, these actions will be described in the third phase, augmenting the decision structure.

The user at this point can ask for a decision matrix for this decision problem. A Decision Matrix, such as shown in Figure 5.7, is the major interface between SIBYL and the user. It displays the goals in the top row and the alternatives in the leftmost column. The value in each cell represents the current evaluation of the alternative with respect to the goal associated with the cell. Initially, each cell displays the value, "unevaluated". As people produce pro and con claims for the alternatives, the values shown in the cells get updated accordingly, as described below.

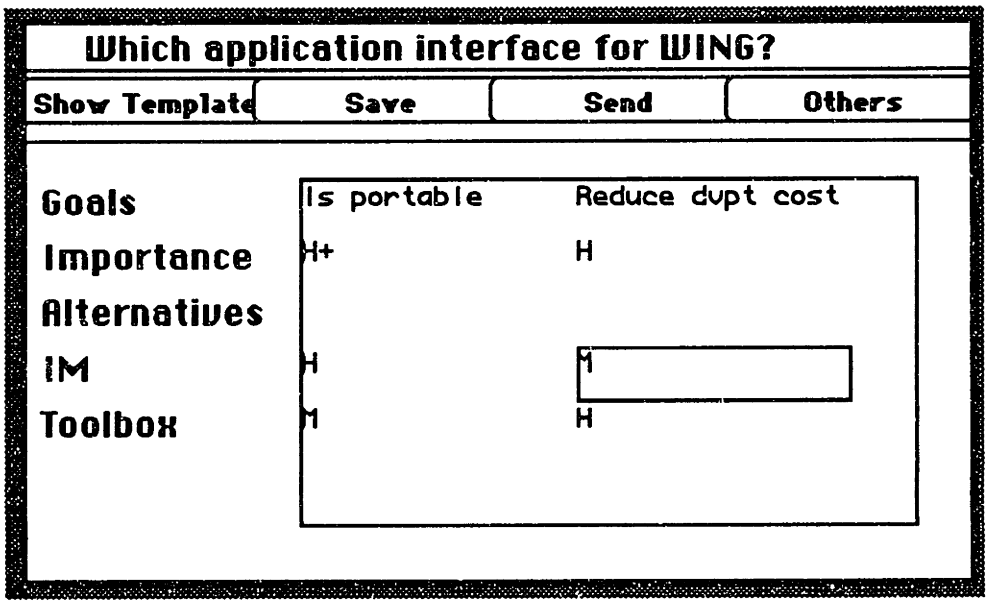


Figure 5.7 A decision matrix displaying the evaluations of the alternatives with respect to different goals

5.4 Releasing the Initial Structure

A decision problem is released when it is made accessible to other users so that they can augment the decision structure by contributing their own knowledge or evaluations about the alternatives, goals, and claims.

There are different ways in which a decision problem can be released.
One way which has been actually used so far is to send the initialized decision as an electronic mail message. SIBYL makes use of the Object Lens feature of sending objects via email. An Object Lens object, including a folder object containing a group of objects, can be sent in a message. These objects are translated into an ASCII representation before they are sent over the net. On the receiving end, Object Lens systems can then parse this representation into the original objects. Because these objects have unique global identifiers and version identifiers, they are loaded only when there are no objects with the same object and version identifiers. (For detailed descriptions of this feature of Object Lens, see Object Lens: User Guide.)

If one decides to make decisions over an electronic network, one would select the decision problem instance and choose the action Send. The system asks whether to send only this object, or all the objects that are immediately linked to the object, or all the objects that are linked to the objects at all (i.e. transitive closure). Since the decision problem has all goals and alternatives linked to it, which in turn can refer to other objects, the third option is commonly chosen so that all the objects connected with the decision problem are sent to the other users. The recipients of these objects can then examine this structure, make their own contributions in the manner described in the next section, and send the augmented structure back to other users.

This method has been used successfully on small-scale decision problems, where there were not many concurrent updates. However, it has one major problem, namely that of maintaining global consistency. Because the decision state that one user sees may be different from that seen by another user at a given moment, an argument that is put forth by one user may be irrelevant or confusing by the time it gets to other users.

Another way of releasing the initial decision, one which would avoid the above problem of consistency, is to store the structure in a shared database and give the other users appropriate access rights. The use of a shared database allows us to take advantage of the concurrency control features of the database in order to help maintain consistency. The database connection to Object Lens, however, has just been implemented and is not quite stable enough yet. Once it is stable, however, SIBYL would adopt this method of releasing a decision problem.

5.5 Augmenting the Decision Structure

Once the decision problem is released, the users contribute to the decision making by:

- relating it to existing objects
- specifying goals and alternatives
- elaborating goals and alternatives
- adding arguments and counter-arguments
- assigning evaluations
- asking questions
- answering questions

The first two processes were described in the previous section. In this section, the rest are described.

Elaborating Goals and Alternatives

As the decision making progresses, the initial goals and alternatives become more specific, and need to be elaborated. In SIBYL, a goal is elaborated by creating its subgoals. For example, the goal "Support many UI styles" is elaborated to mean "Implement direct manipulation" and "Support form-based interface easily." To create subgoals for a given goal, one brings up a goal browser, such as one shown in Fig. 5.8, which shows all the goals for a given decision problem. When one clicks on one of the goals, a pop-up menu appears with a set of actions that can be performed on that goal. One of the actions is Create a Subgoal, which, when executed, will bring up a template editor containing a new instance of the goal, and automatically link it to the goal clicked via a subgoal relation. When editing is finished and the new goal is saved, the goal browser is automatically updated to include the new goal related to the goal clicked via a subgoal link.[7]

Another action in the pop up menu, Argue about Relations, allows us to argue about whether something should be a goal at all. When the action is chosen, SIBYL presents a pop-up menu displaying a list of all the relations between the object selected and other objects (remember that relations are first class objects). To argue whether something

[7] If one also believes that these subgoals are exhaustive in the sense that satisfying all of hem is equivalent to satisfying the parent goal, that they are disjunctive in the sense that satisfying one of them is equivalent to satisfying the parent goal, or that they are mutually exclusive, or mutually independent, then one can specify these different relations by relating the subgoals to the parent goal via a Group object and specify these relations as one of the attribute of the Group object. This information about the relationship among the subgoals is used by the plausibility management later when the plausibility of the claims are propagated through these *Is A Subgoal Of* relations.

should be a goal at all, we choose the relation *Is A Subgoal Of*, which links the goal object in question to its parent goal (possibly the decision problem object). This relation -- the claim that the goal is a subgoal of the goal that it was linked to -- is displayed in a template editor containing the instance of that relation; one can now support, deny, or qualify that relation in the manner described in the next subsection on adding arguments.

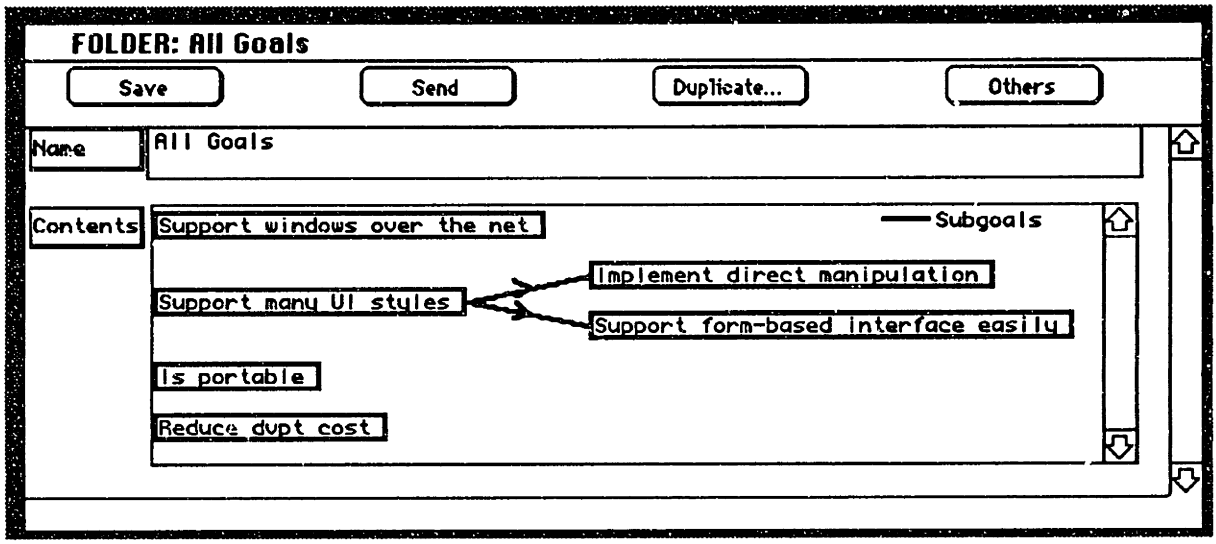


Figure 5.8 A goal browser displaying all the goals and their relations

So far, I have described how to add goals, additional alternatives, elaborate existing goals, and argue whether they should be goals at all. Alternatives can be elaborated and argued about in a similar manner via the alternative browser, which displays all the alternatives for a given decision problem.[8]

[8] Another way of adding, elaborating goals and alternatives is by bringing potentially useful objects from past decisions. The precedent manager helps one do so and it is discussed in Section 6.1.

Adding Arguments and Counter-Arguments

In SIBYL, one evaluates an alternative by arguing about how well it achieves each of the goals specified so far. The user can do so by bringing up an argument browser associated with the alternative and the goal in question. For example, if the argument is that an interaction manager interface is good for portability because it provides a strict separation between abstraction levels, then one would click on the cell which is in the intersection of the alternative [IM] and the goal [Is portable]. Figure 5.9 shows an argument browser as well as the cell that the user clicked on to bring up the browser. This claim that an alternative achieves a goal is automatically generated by the system for each alternative whenever a goal is created, and an argument browser for each cell in the decision matrix initially contains this claim for people to argue about.

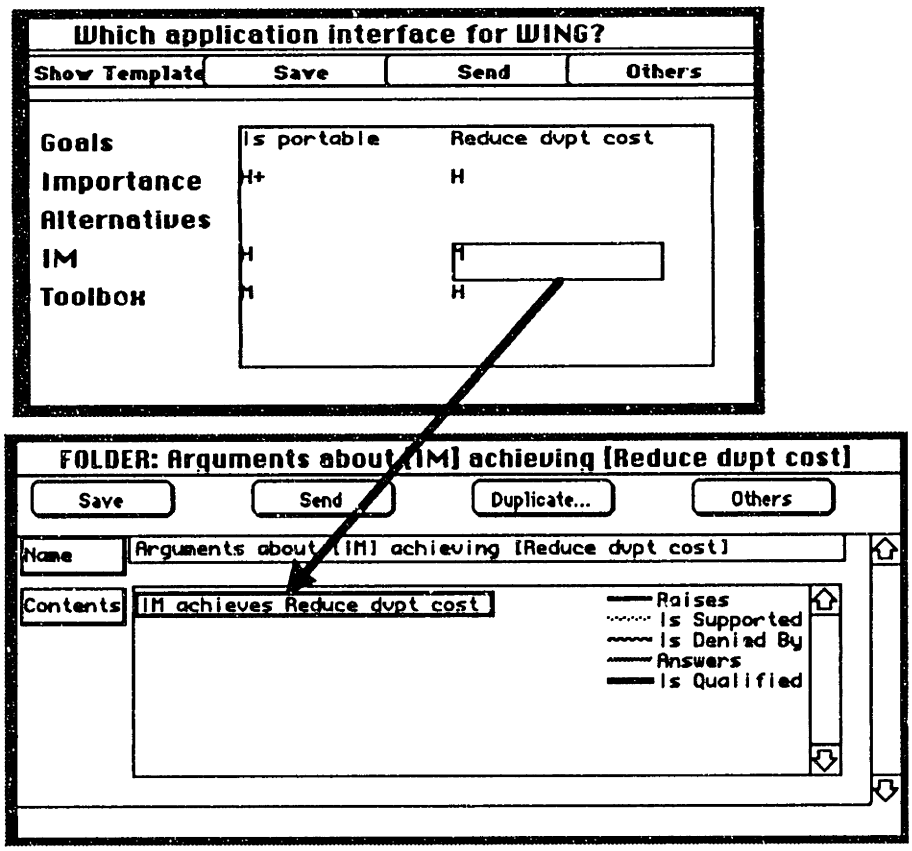


Figure 5.9 An argument browser displaying arguments for the evaluation of the alternative [IM] with respect to the goal [Reduce dvpt cost], shown in the decision matrix above.

Users express their pro and con arguments as claims supporting or denying this Achieves claim; hence, the evaluation of the claim represents the measure of how well the alternative is achieving the associated goal. In SIBYL, the user can always mouse-click on an object and get a menu of all possible operations that can be performed on the object. The user can mouse-click on this initial Achieves claim and get a menu displaying possible actions such as Add A Supporting Claim, Add A Denying Claim, Add A Question, and Add A Qualification. When the user chooses Add A Supporting Claim, for example, a template editor containing the new instance of Claim is brought up and this new instance is automatically linked to the original claim through an instance of the Supports relation. When the user chooses Add A Supporting Claim, SIBYL displays a template editor containing the new claim instance, and links this new claim to the original claim via a Supports relation. Figure 5.10 shows the argument browser after many people's contribution. An argument browser is in fact a window into a particular portion of a decision graph, e.g. the region bounded by a heavy box in Fig. 5.11.

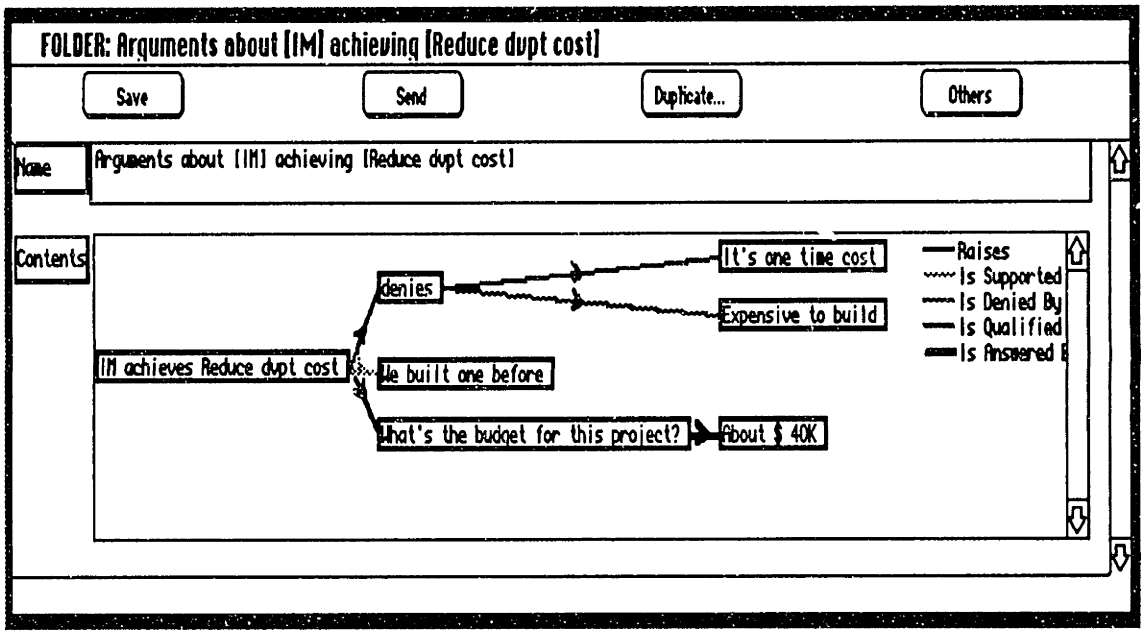


Figure 5.10 The argument browser with arguments and counter-arguments added.

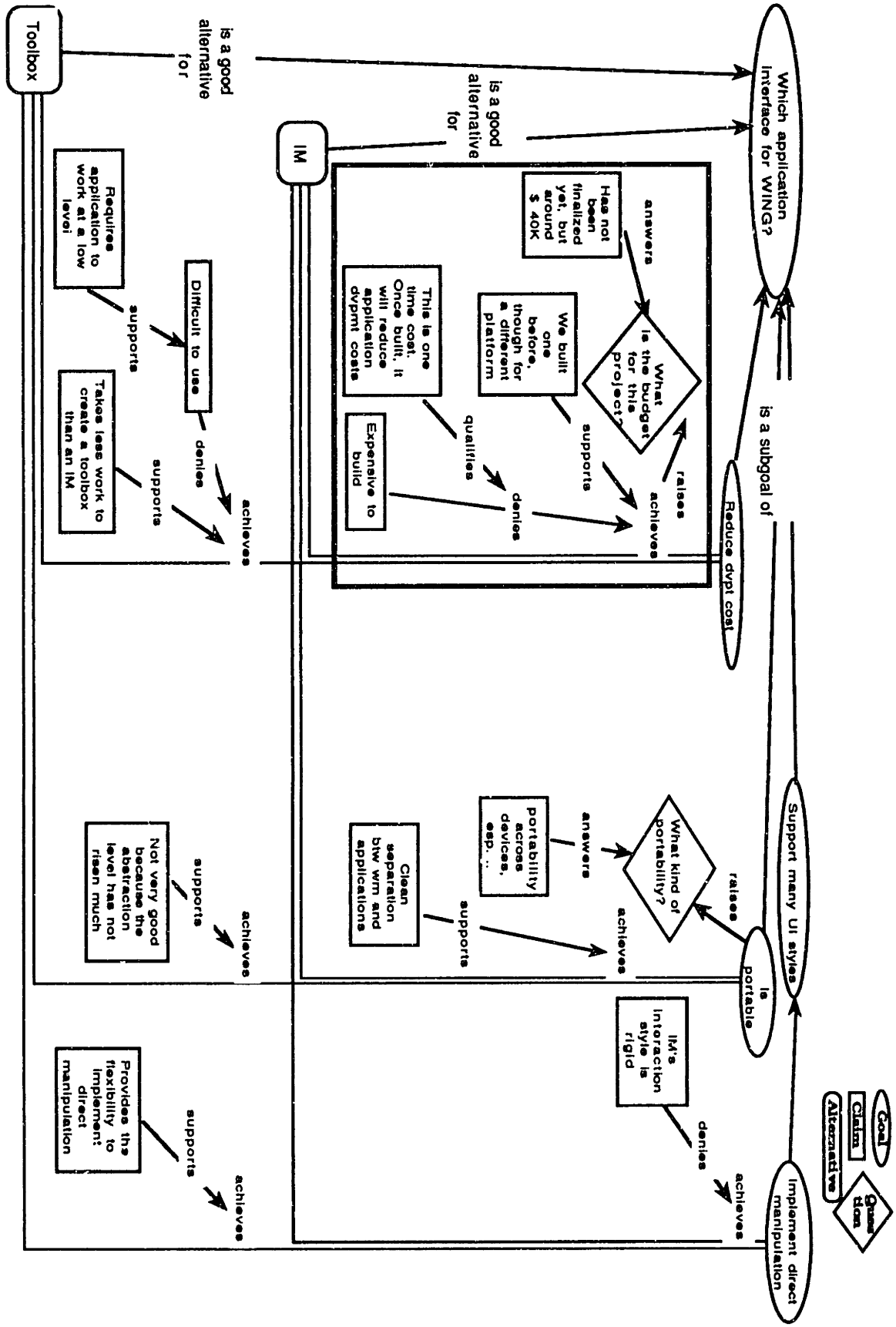


Figure 5.11 The region of the example decision graph, bounded by a heavy box, which is displayed in the argument browser in Fig. 5.10

Asking and Answering Questions

Whenever the user wants more information or clarification concerning an existing object, he can ask a question about it. He does so by, again, mouse-clicking on the appropriate node in the argument browser and by choosing from the pop-up menu the appropriate action, in this case Raise A Question. This action creates a new instance of a question, links it to the object clicked via a *Raises* relation, and updates the argument browser. Later, when another user wants to provide an answer to this question, that user clicks on the question and choose from the menu the action, Answer A Question. This action creates an instance of *Claim*, links it to the question via an Answers relation, and updates the browser.

Assigning Evaluations

When the arguments are added as described above, the decision matrix is updated to reflect the changes in evaluation that might result from the addition of the arguments. Automatic updates by the system are possible, but there are still many problems that would have to be resolved. Hence, currently, the update is done manually by the users in the following manner.

A user, usually after making his own contribution, would examine the argument browser associated with a given cell of the decision matrix. The cell already shows some evaluations, if only it is "unevaluated". The cell may display a list of evaluations because different users may have different evaluations of the same set of arguments. Each evaluation can be:

- a string with no other associated information, e.g. "unevaluated" "unanswered questions",
- a list of simple keyword evaluations and their creators, e.g. (HIGH, Susan)
- an instance of a claim with its usual attributes like creator, creation time, supporting claims, denying claims, and others.

The way in which the user would express his evaluation would depend on what he has done:

- If he has performed actions (e.g. answered a question, made the first evaluation) that clearly annul a current evaluation (e.g. "unanswered question" or "unevaluated"), he would just take appropriate actions on these evaluations, e.g. delete "unanswered question" if answered or replace "unevaluated" with his own evaluation. In expressing his own evaluations, he can either use a text string (e.g. "H"), a list of keywords (e.g. (HIGH, Susan)), or an instance of a claim, (e.g. [High]), with the creator field filled with his name (Fig. 5.12). One would usually use a text string or a list of the keywords until there is a need to make it into a claim (when it becomes the subject of an argument, for example).

- If he disagrees with his own earlier evaluation, either because of new arguments or a change of mind, he can replace it with his new evaluation.

- Otherwise, he simply inserts his own evaluation in the list. If he does not understand or disagrees with others' evaluations, he can ask a question or argue about the evaluations. To do so, he might have to make some string-based evaluations into claim instances.

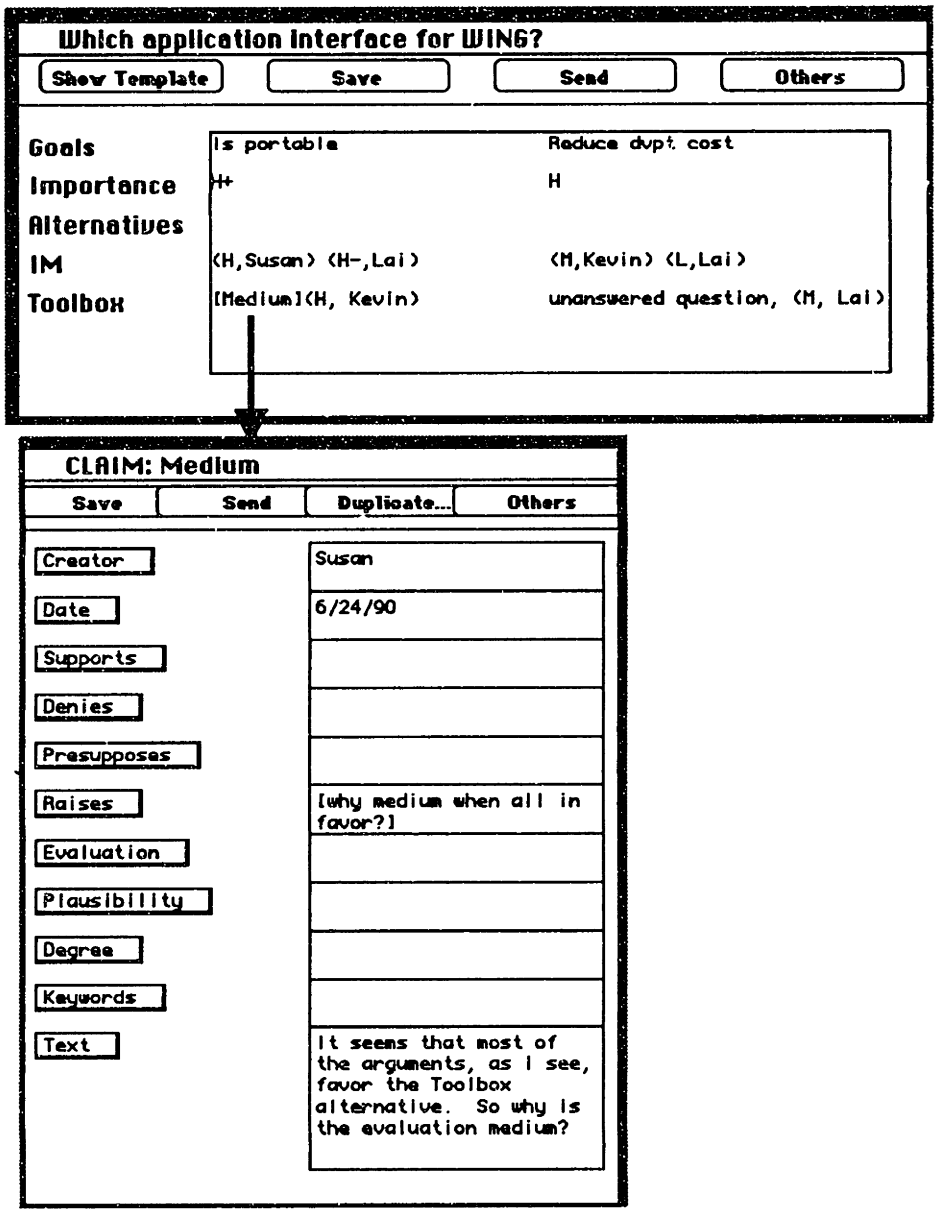


Figure 5.12 A formalized evaluation measure. The evaluation [Medium] shown above is an object with its own attributes, which can be edited and examined.

A decision matrix shows only goals at a given level, that is a goal and its subgoals do not appear in the same matrix. The subgoals of a given goal are shown in the submatrix associated with the goal. The user examines the submatrix for a given goal by mouse-clicking on the goal link in the decision matrix and choosing from the pop-up menu the action

Show the Subgoal Matrix.[9] A decision submatrix for the goal [Support many UI styles] is shown in Fig. 5.13.

Hence, when there are subgoals, there is an additional evaluation process. When the cells in a submatrix have been assigned evaluations, one goes up one level and should assign -- on the basis of the evaluations on the submatrix -- an evaluation for the appropriate cells in its parent matrix that the submatrix is associated with.

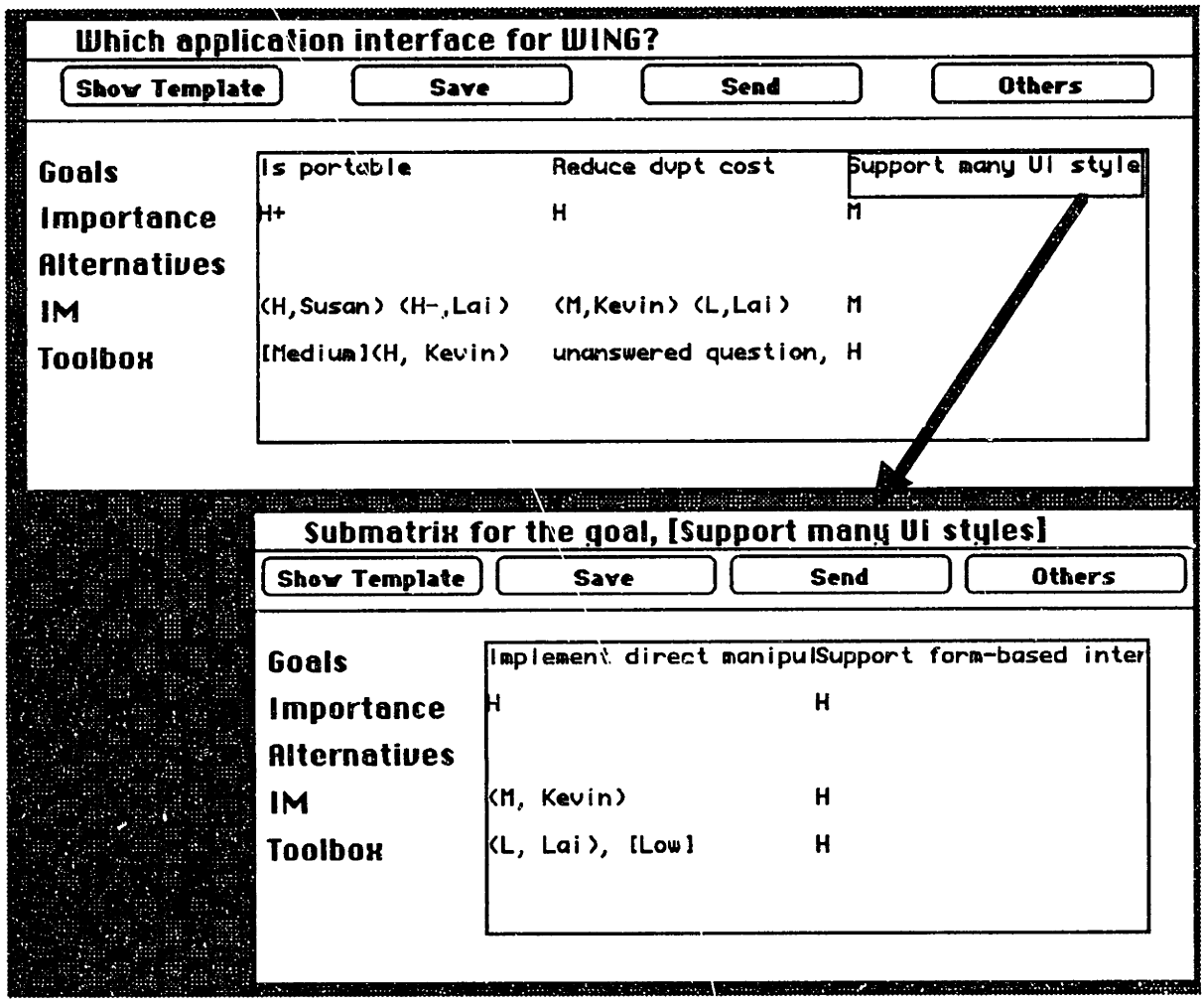


Figure 5.13 A decision submatrix which shows the rationales for the evaluation of an alternative with respect to a goal in terms of its evaluations with respect to the subgoals, [Implement direct manipulation] and [Support form-based interface]

[9] This feature is implemented in the Interlisp implementation,.

5.6 Making the Decision

SIBYL does not automatically make the decisions; it only helps the user make a decision. SIBYL assumes the existence of somebody, human or potentially a computational agent who can examine the decision structure that have been elaborated so far and assign final evaluations. For example, if the current evaluation of the cell is '(H, Kevin), (L, Lai)', then an arbiter is needed to assign the final value, presumably based on the credibility or reliability or expertise of the source for the differing evaluations.

Once the evaluations have been assigned to the cells and the questions that need to be answered have been answered, this decision maker or arbiter determines the final evaluation for each of the cells in the matrix based on the list of current evaluations as well as the arguments (shown in the argument browser\ responsible for these current evaluations. If there are submatrices, this process starts at the bottommost submatrix and recurs up until it reaches the topmost matrix. The decision maker decides based on the evaluations shown in the topmost matrix at the current level .

Once the decision is made, one would record the outcome and the reasons for the outcome by creating an instance of the type *Decided* and placing a link to it in the Status field of the decision problem. *Decided* is a subtype of *Status*, which is used to represent information about the current state of a decision problem. The Chosen Alternative attribute of *Decided* contains typically the chosen alternative or a description of whatever closed the discussion about the decision, for example a description to the effect that the decision is no longer relevant. The Rationales field of *Decided* points to those objects that were mainly responsible for the final decision. For example, Fig. 5.14 shows that the decision was made to use interaction manager as the WING application interface, and that the major reasons for this decision were the arguments about portability and about reducing

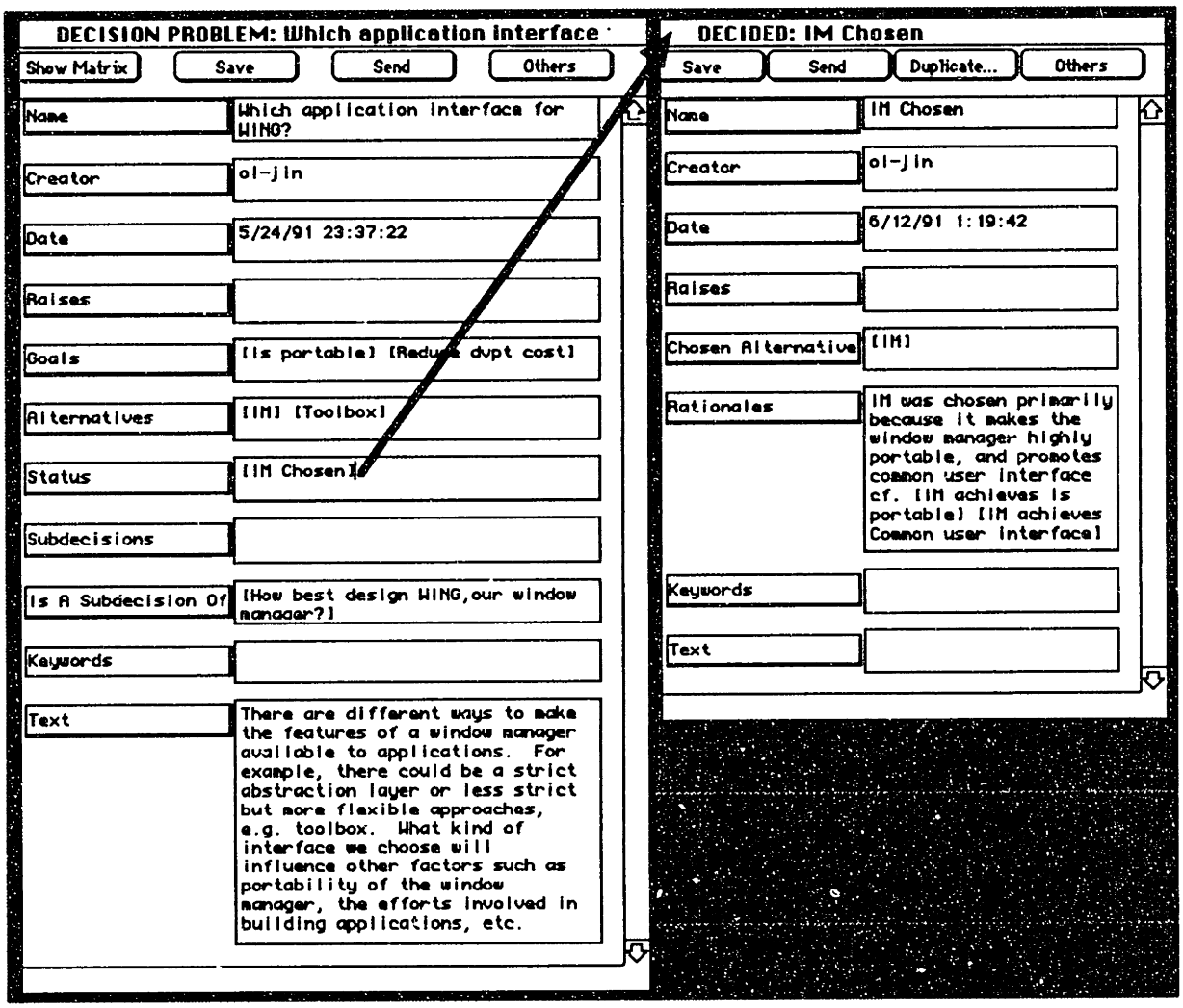


Figure 5.14 *Decided*, a status for a decision problem, shown as a formal object with its own attributes that can be edited and examined

application development cost both in favor of the interaction manager. It could refer to itself as the reason if all the arguments in it are judged to have been equally important.

5.7 Evaluating the Outcome of the Decision

Later, if the success or the failure of the decision is ever known, one can come back to the decision and fill in the Evaluation field of the decision problem. The values one can fill in can be text strings such as Utter Failure, but more instructive ways to do so involve

creating an instance of *Claim*. As shown in Fig. 5.15, one would use this claim to record why one believes that the decision was a success or a failure, and which parts of the decision were responsible. This way, others can also argue about this evaluation as a claim.

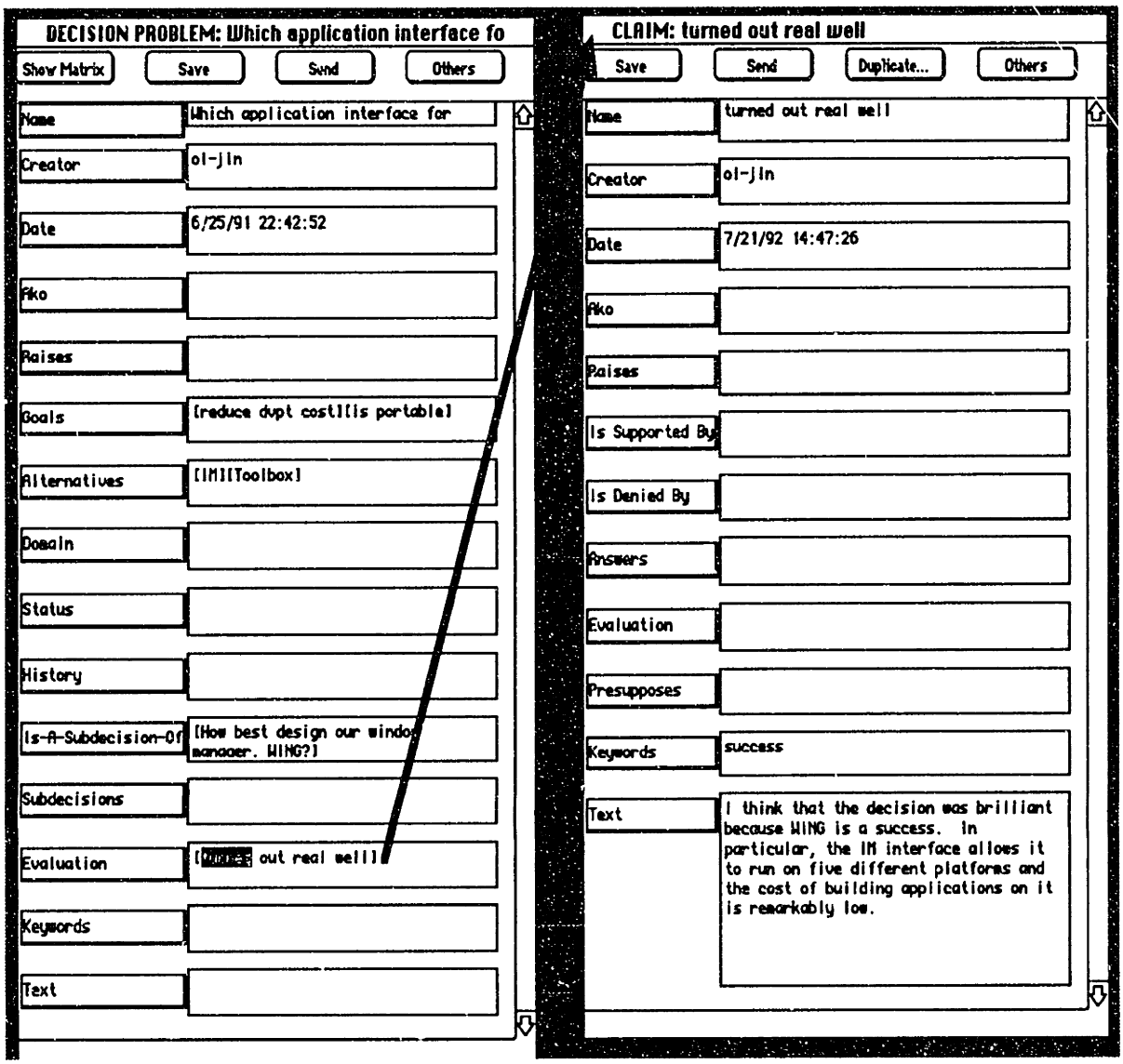


Figure 5.15 An evaluation of the outcome of a decision problem, [turned out real well], entered as a formal object with its own attributes can be edited and examined. The evaluation field was not shown in the earlier figures because it was hidden, as any field can be until needed.

Another way is to create special *Success* and *Failure* subtypes of *Claim,* representing the claim that it was a success or that it was a failure respectively, and use an instance of these new subtypes to record the reasons for the evaluation. These types may have their own attributes, like Rationales, which point to the objects that one thinks are responsible for the success or the failure of the decision. Creation of these types would allow the system to more easily determine whether a decision was a success, and give the system more basis for computational operations on these evaluations. The types, such as *Success* or *Failure*, are not built-in types of DRL, though, because presumably different applications would want to categorize these evaluations differently.

Chapter 6

Computational Services based on DRL

Thus far, I have shown how rationales can be captured using SIBYL. The goal of this chapter is to show what can be done with the captured rationales. I present the computational services that use the captured rationales to support decision making. There are three major services that SIBYL provides: the management of precedents, dependencies, and viewpoints.[10] The precedent manager helps the user to retrieve rationales from past decisions that may be useful for the current decision. Because a major motivation for this research is the desire to reuse rationales, the precedent management has received most attention (Section 6.1). The other two services, dependency and viewpoint

[10] There was an attempt to develop an evaluation manager that is responsible for automatically merging and propagating evaluations. The problems encountered in the attempt are briefly described later.

managements, deal with two kinds of needs that were found to be of critical importance from the experience of representing and managing the rationales in example domains: maintaining dependencies on the one hand and keeping track of multiple decision states on the other. The dependency manager helps the user to propagate the consequences of changes and maintain consistency across decisions (Section 6.2). The viewpoint manager allows the user to create multiple viewpoints and compare them (Section 6.3). The rest of the SIBYL services, such as monitoring and sharing objects through email, are grouped and described at the end (Section 6.4).

6.1 Precedent Management

One of the major motivations for this research is the desire to reuse rationales. This makes the precedent manager the most important of the services.

The job of the precedent manager (PM) is to bring into the current decision useful knowledge from past decisions. There are two ways in which the user might want to retrieve past decision rationales. First, he might have something specific to look for, e.g. any argument about how well the toolbox approach promotes a common user interface. For this case, SIBYL allows the user to specify a partial structure and retrieves any structure matching the specified structure. This case is described in the first subsection. Alternatively, the user might not have anything specific in mind, but just want to know if there may be anything relevant in past decisions. In this case, SIBYL has to decide which of the past decisions might contain useful knowledge. This case is discussed in the second subsection.

6.1.1 Specific Retrieval Request

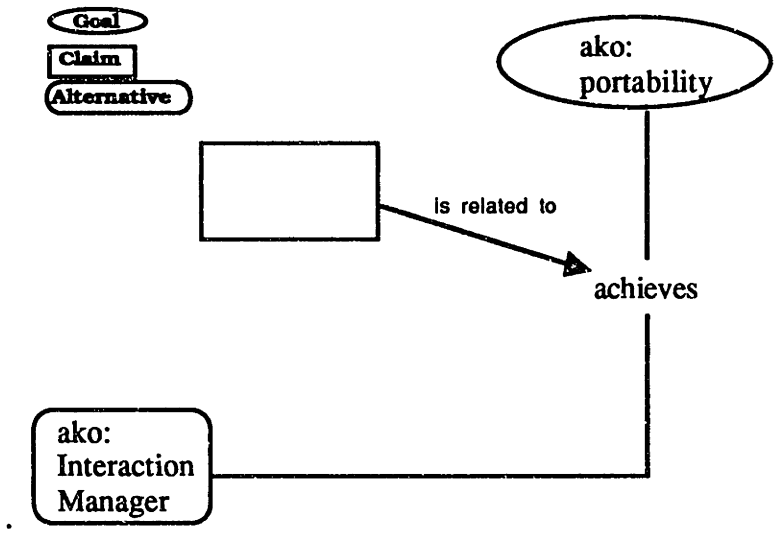
Often, the user wants to find a specific type of information from past decisions. For example, he might want to know whether a certain kind of alternative has been considered in the past, whether a question about a particular product has been answered before, or whether there is any argument evaluating a certain type of alternative with respect to a given

type of goal. In this case, the user supplies a partially specified structure composed of SIBYL objects as a query.

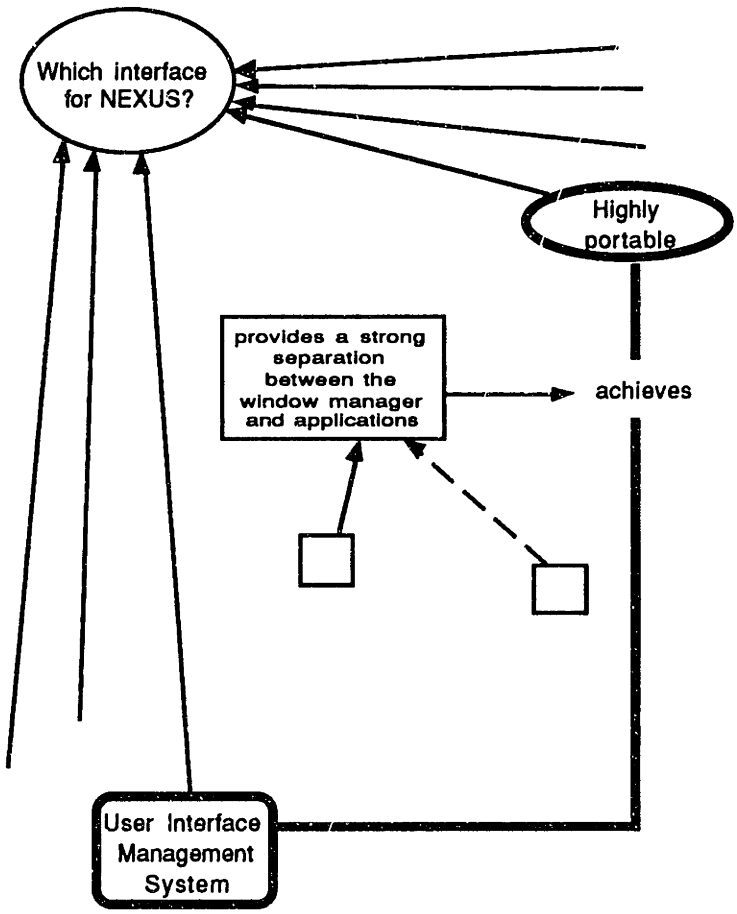
A query consists of a set of descriptions and their relations. A description is a template of a given type with its attributes partially filled in. Figure 6.1a shows a graphic representation of a partial structure which consists of three descriptions (of type *Claim*, *Alternative*, and *Goal*) and their relations. This partial structure specified as a query would, for example, retrieve any claim which is about an Achieves relation between an alternative of type [Interaction Manager] (specified by the specified attribute value, ako: interaction manager) and a goal whose type is [Portability]. Figure 6.2 shows actual user interface in which this partial structure is specified.

Figure 6.1b shows a retrieved structure. It matches the partial structure specified in Fig. 6.1a because [User Interface Management System] is an alternative which is a kind of [Interaction Manager] and [Portability across Hardware] is a kind of [Portability]. The retrieved structure also contains any object that is immediately linked to the matched structure. For example, the claim [Has limited application semantics] is also retrieved because it is attached to the Achieves relation that matches the Achieves relation specified in the query. Only those objects immediately linked, i.e. only one level deep, are retrieved. However, each retrieved object also has pointers to the other objects that are immediately linked to it, so that the user can retrieve objects more distantly related to the original partial structure by following the links.

The implementation for this case is straightforward with the Object Lens rule system. Figure 6.2 shows the actual user interface through which this partial structure is, i.e. as the if-part of a rule whose condition consists of a nested partially filled descriptions.



(a) An example of a partially specified structure



(b) A retrieved structure that matches the partial structure in (a).

Figure 6.1 A partially specified structure used to retrieve potentially relevant rationales

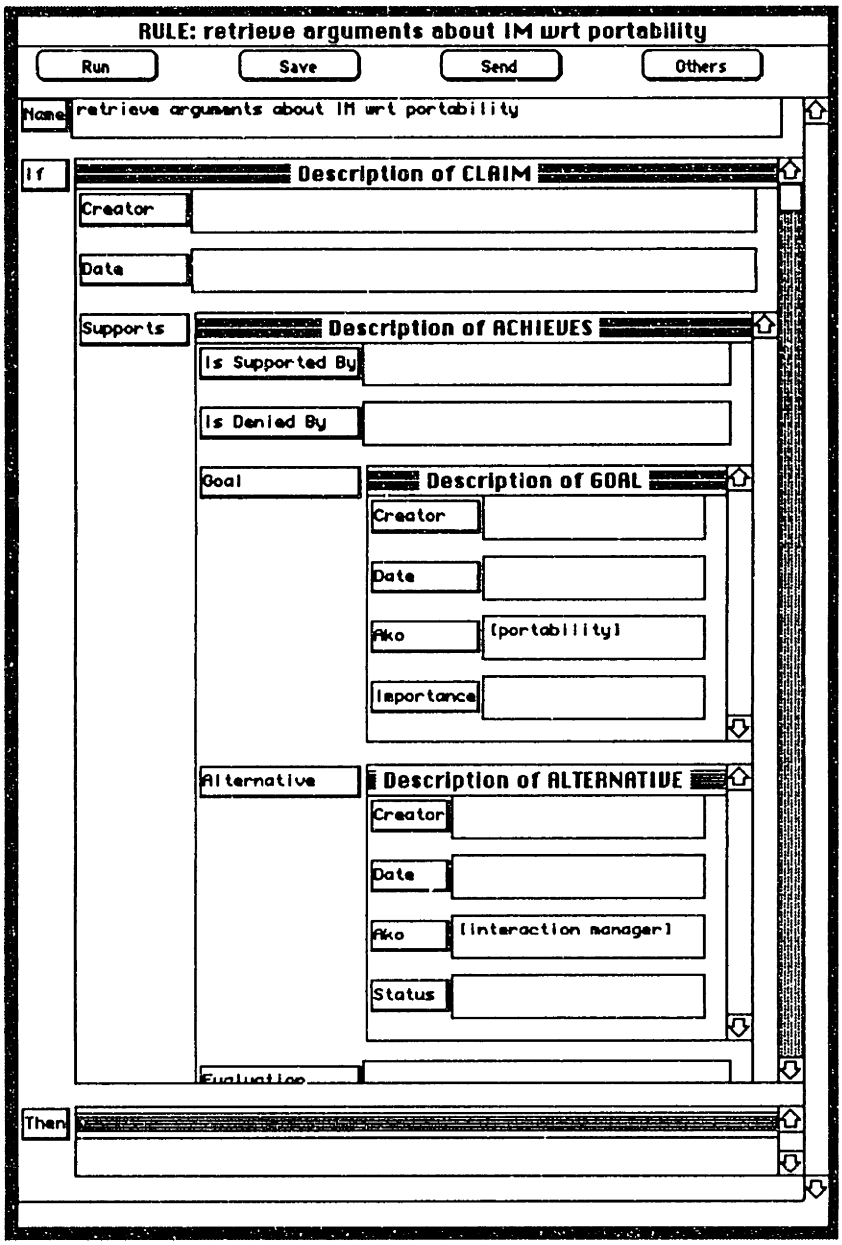


Figure 6.2 The actual user interface used to describe the partially specified structure shown in Fig. 6.1a.

The actions specified in the then-part of the rule is performed on all the instances that match the nested descriptions. Hence, in the example, any claim which support any instance of the *Achieves (A, G)*, where A is a kind of [Interaction Manager] and G is an instance of the goal [Portability] would be retrieved and moved into a folder called [All the Claims

evaluating an Alternative of type, Interaction Manager, with respect to a Goal of type, [Portability].[11]

6.1.2 General Retrieval Request

The user can also request retrieval of any relevant information from past decisions without providing a partial structure to match. Figure 6.3 shows what the PM needs to do in this case. To bring in useful knowledge from past decisions, we need to *find* those decisions that might contain useful knowledge, *extract* the relevant pieces of knowledge from them, *adapt* them to the current decision, and *link* them to appropriate places in the existing body of knowledge. The user can then examine the transferred knowledge and determine what its effect on the current decision should be.

Selecting past decisions to look at

To determine which past decisions to look at for potentially useful knowledge, the PM does the following:

(1) *Define a similarity metric* with which to measure the similarity between decision problems,

(2) *Rank the past decision problems* using the metric, and

[11] This implementation of rules has many limitations. The rules do not support variables nor conjunctive conditions within a field. Disjunctive condition is specified by creating multiple rules. In one of the implementations, all of these features are supported, but only by way of low level interaction with the user (e.g. use of Lisp constructs).

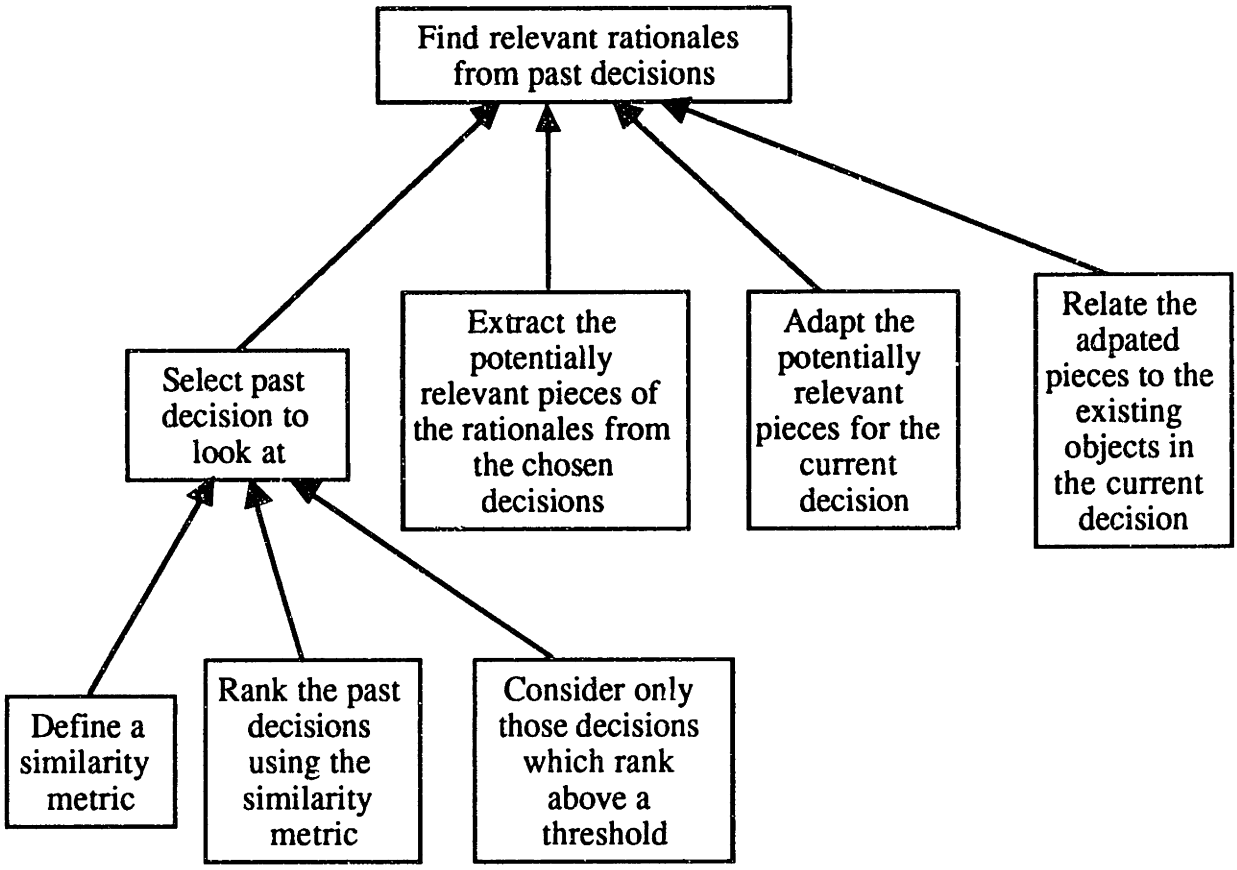


Figure 6.3 Tasks involved in retrieving useful rationales from past decisions.
A set of subtasks converge to a single task.

(3) *Consider only those decisions which rank above a threshold* (set by default overridable by the user) and apply the next stage algorithm for extracting relevant pieces.)

Each of these substeps is described below.

Defining a similarity metric

SIBYL uses the number of shared goals as the similarity metric between decision problems. Underlying this choice is the intuition that the more goals two decision problems share, the more knowledge they are likely to contain that is potentially relevant to each other. For example, the alternatives and the claims that evaluate these alternatives in a

decision are likely to be relevant to another decision to the extent that the two decision problems share similar goals.

Using the similarity metric to rank the past decision

To rank past decisions using the similarity measure based on the number of shared goals, the PM needs to know what it means for two decision problems to share goals. If we have two decision problems, one with the goal "Easy to use" and the other with the goal, "Ease of use," do they share a goal? The answer is, of course, that it depends. Most likely, there would be some differences between the two goals. Whether these differences can be ignored depends on the particular purpose at hand, i.e. for the purpose of making the current decision. Currently, the PM leaves this decision to the user, but helps the user make the decision by narrowing down the potential candidates for matches. Described below is the algorithm that the PM uses to look for potential matches to suggest.

The PM retrieves potential matches for a current goal with the use of a goal lattice. I defer the explanation of how a goal lattice gets created until after I explain its use in deciding which of the goals are shared across decision problems. A goal lattice shows the goals in a given task domain. Figure 6.4 shows some of the goals in the task-domain of designing an optimal application interface. This goal lattice shows two kinds of relations among goals: specialization and example. The *specialization* and *examples* correspond to the usual subtype and instance distinctions.[12] That is, a goal G2 is a *specialization* of another goal

[12] The terms, specialization and example, are used because users of SIBYL are often not familiar with the subtype and instance distinctions.

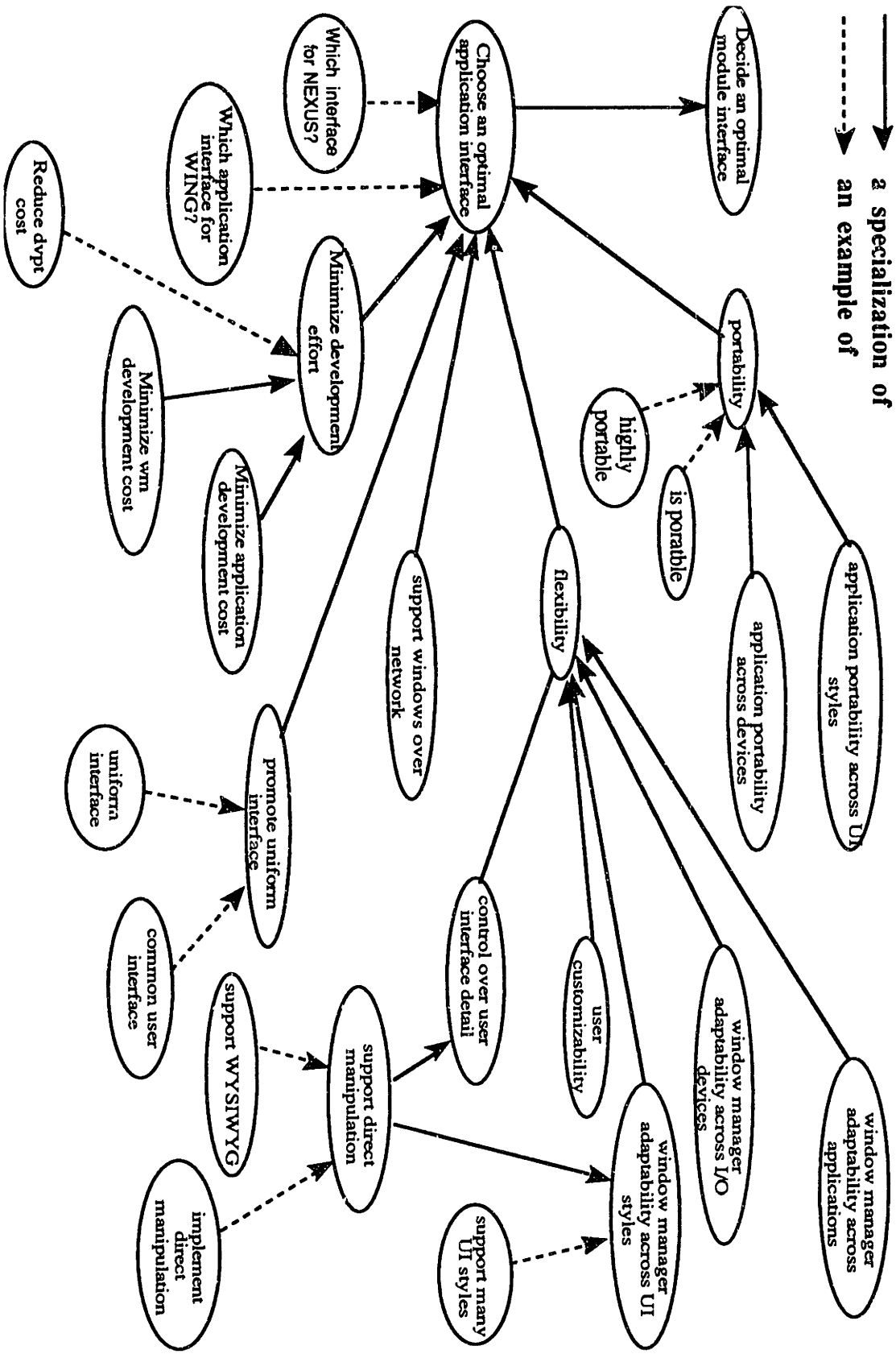


Figure 6.4 A goal lattice displaying the goals that have been created for the decision problems of the type "Choose an optimal application interface."

G1 if both G1 and G2 are types, and all the instances of G2 are instances of G1. For example, "Choose an optimal application interface" is a specialization of the goal "Design an optimal module interface." A Goal G2 is an *example* of G1 if G1 is a goal type and G2 is an instance of G1, i.e. is in fact used in some decision problem. A *goal type*, on the other hand, is never used directly in a decision problem.

Given a goal lattice, the PM decides which past decisions to retrieve in the following way. For each goal in the current decision, find other examples of its goal type in the goal lattice. Then for each of these examples, ask the user whether it matches the current goal. In Fig. 6.4, the current goal, [Which application interface for WING?], and [Which interface for NEXUS?] are both examples of the goal type [Choose an optimal application interface]. SIBYL presents each of the examples found to the user as a potential match for the current goal, which then confirms or denies the match. Each of these matched goals points to the original decision problem in which it appeared. Using this information, the PM can rank these past decisions by the number of matched goals. Figure 6.5 shows a precedent with three of its goals matched to the current goals (those shown in heavier frames).

To ensure the success of this heuristic based on the goal lattice, the lattice has to be managed carefully. For example, if a goal type in the lattice is too general (e.g. [Design good software]), its examples may not reflect similar concerns and may not be good matches (e.g. [Design a good window manager], [Design a good database]). In the following, I discuss how the lattice is to be constructed and maintained.

The goal lattice is created partly top-down and partly bottom-up. If the task-domain has a taxonomy of goal types that is well-agreed on, then it can be used as an initial structure for the goal lattice. For example, Fig. 6.6 shows a taxonomy of the goals in software engineering [Boehm 1981]. These goal types can then serve as basis for adding more

specialized goal types and examples in the manner described below. If such a taxonomy is not available for the domain, the lattice can start with a single node representing the top level task, e.g. [Design a window manager], which can then be specialized. The lattice that grows in the process can then serve, in turn, as a basis for producing a taxonomy for that domain.

When the user posts a goal for a decision problem, he is required to use an existing goal type or create one by specializing an existing goal. For example, if the user wanted "direct manipulation" as a goal, he would search the lattice that has been built so far, decide which path to go down until he comes across a goal (e.g. [Implement direct manipulation]) which corresponds to the requirement of direct manipulation or he reaches a leaf of the lattice without finding anything. In the first case, he would create an instance of the found goal type and post it as a goal in the decision problem in question. In the latter case where he does not find any existing type to be suitable, he specializes one that is as specific as possible yet more general than [Support direct manipulation]. He then uses an instance of it for his purposes.

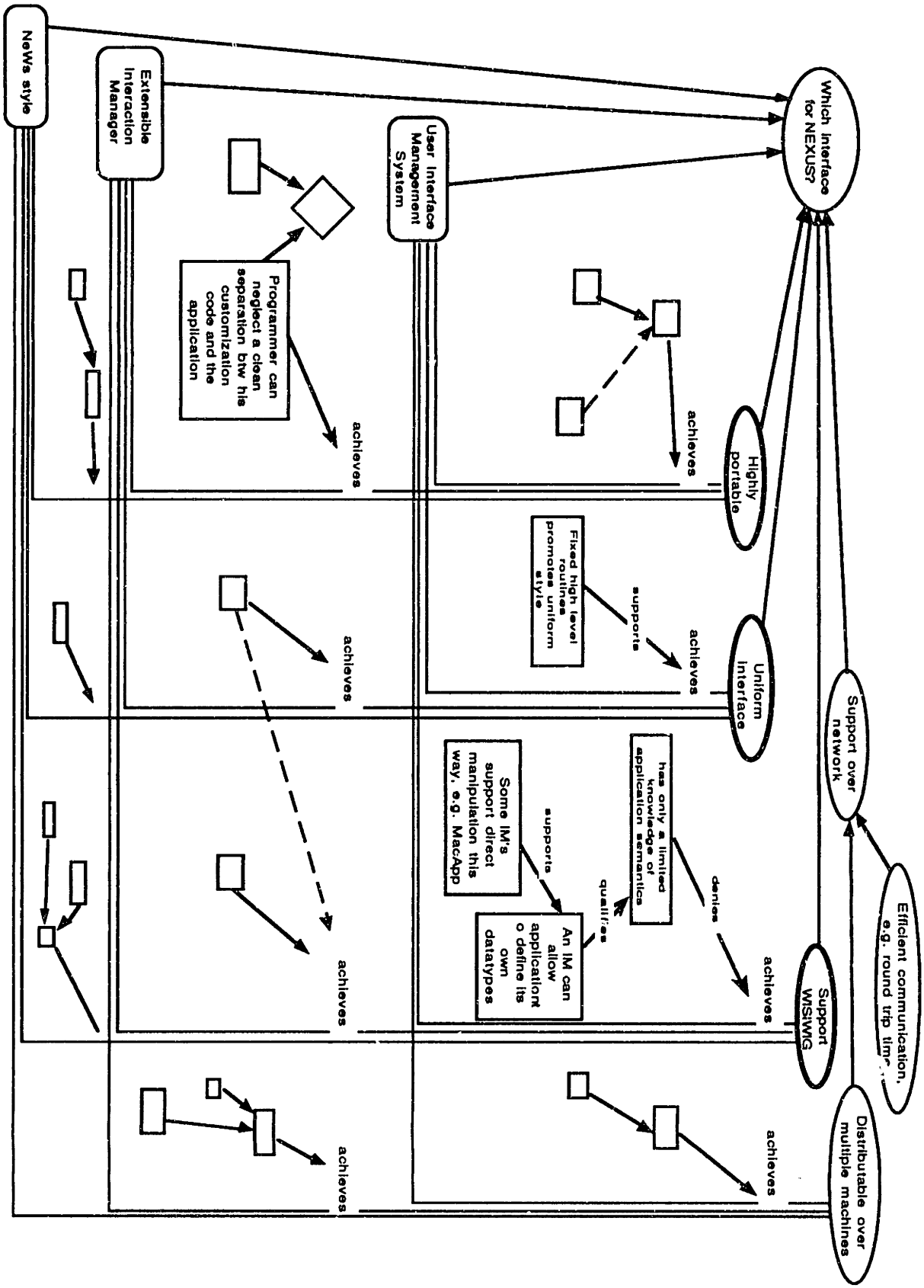


Figure 6.5 The decision graph for a precedent. Only those nodes used for the discussion are shown in detail.

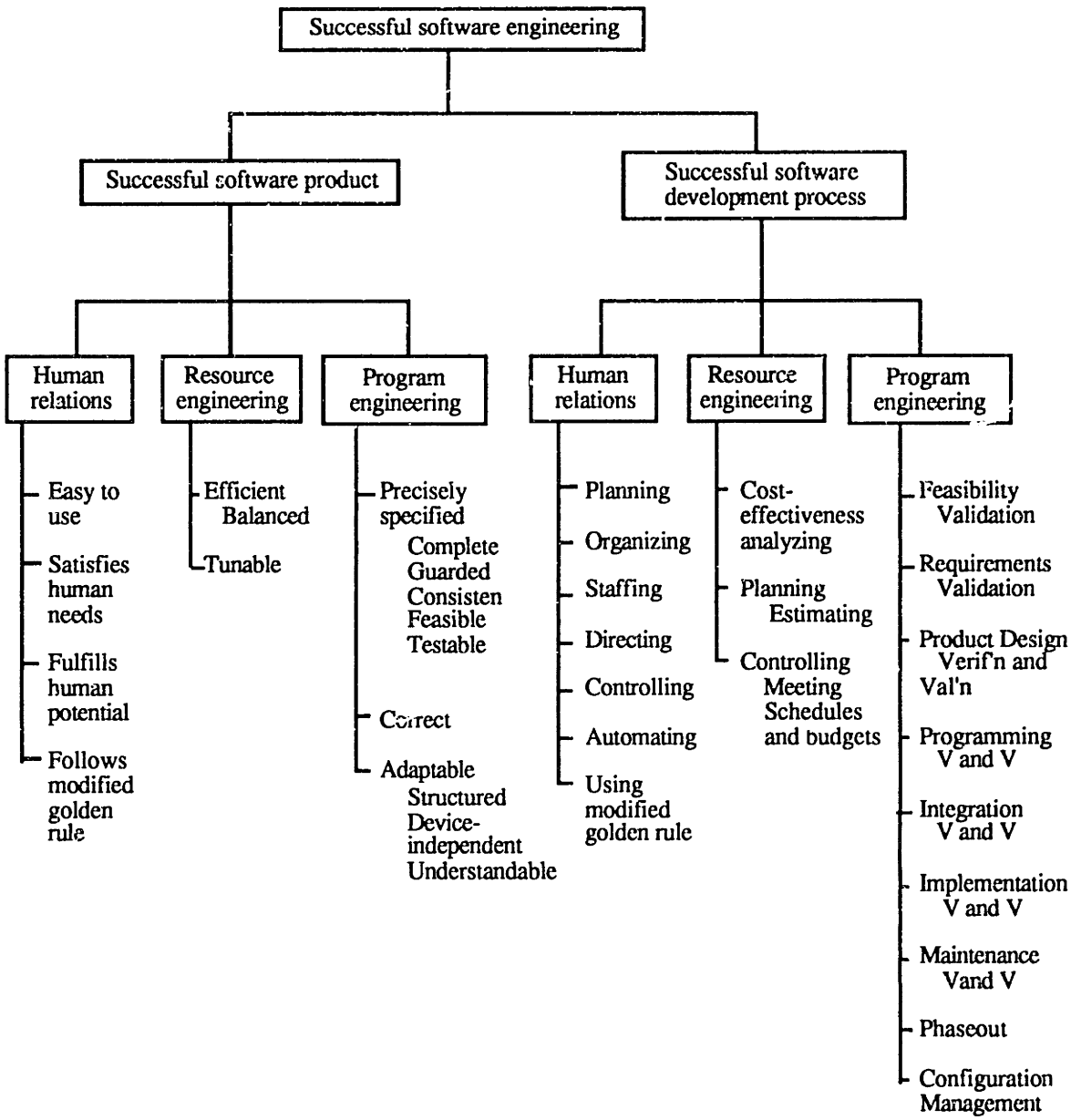


Figure 6.6 High level goals in software engineering (from [Boehm 1981], Fig 3-5)

The goal lattice growing this way can be populated with random goal types. In order to increase the chance of finding potential matches as well as help the user find or create his

goal in this lattice, the goal lattice is maintained by a supervisor knowledgeable in the domain.[13] The job of the supervisor is to monitor the lattice and reorganize it once in a while by creating new types that generalize or specialize existing types and redistributing them.[14] With appropriate supervision of the goal lattice, the heuristic at least provides an initial filtering mechanism for suggesting most plausible candidates, which can then be confirmed by human users.

The goal lattice is not the only way in which potential matches can be found. A large body of research, ranging from precedent-based learning [Winston 1982], analogy [Faulkenhainer 1989; ; Hall 1989; Winston 1980], case-based reasoning [Kolodner 1988; Riesbeck & Schank 1989], has proposed various methods for finding good matches. However, these methods require much domain-specific knowledge or at least knowledge more formalized than what is available to SIBYL. Given that a goal of SIBYL is to explore the benefits of a system which is useful whose input is entirely informal but becomes more powerful as more knowledge becomes formalized and accessible, SIBYL could not exploit these methods. In the section on future research, I discuss ways in which SIBYL can be given more knowledge and ways in which these matching methods might be incorporated into the PM.

---

[13] There is currently one goal lattice for a given domain such as software design or hardware choice. A goal lattice is globally shared across decisions within its domain. That is, the goals posted for all the decisions are all created as instances of the goal types in the same lattice. When the decisions are of a similar type, say those involved in designing a window manager, the goals that accumulate in the lattice show all the requirements that the designers of a window manager had over time. This lattice, therefore, can be used as a checklist for a designer of a new window manager. When the decisions cover a wide range of topics, however, several goal lattices are needed so that each lattice does not become a mixed bag and its size does not get overwhelming. The lattices themselves would have to be indexed in some way for the user to know which lattice to look at. The current research does not address these issues yet.

[14] The use of classifier, such as (Schmolze & Lipkis 1983), would have alleviated the need for human intervention. If we can assume the formalization of domain knowledge as well as the goals specified, then we can in fact exploit such automatic classification here. However, without such formalization, goals often involve informal descriptions , which make the use of automatic classifier not possible.

Considering only those decisions which rank above the threshold

Once the past decisions are ranked using the heuristic described above, the PM applies the next stage algorithm for extracting relevant pieces to each of them from the most highly ranked as long as it is ranked above the threshold. The threshold is specified in terms of the number of shared goals, and set by the user.

Extracting and Linking the relevant pieces from retrieved decisions

After the past decision problems are ranked based the number of matched goals, the pieces of their rationales potentially relevant for the current decision need to be extracted.

There are four kinds of potentially reusable rationales captured in DRL:[15]

- Goals, their relations, and arguments about them (type I)
- Alternatives, their relations, and arguments about them (type II)
- Evaluation of alternatives with respect to goals, arguments about them (type III)
- Decision Problems, their relations, and arguments about them (type IV)

Figure 6.7 shows the DRL structures that capture these rationale types. These rationales are potentially useful for the current decision because they tell us about the requirements

---

[15] This categorization of the reusable rationales is based on the analysis of the components of decision rationales presented in Section 8.2. This analysis distinguishes five components of decision rationales: the criteria space, the alternative space,the evaluation space, the issue space, and an argument space behind each of the four spaces. The Type I through IV rationales correspond to the rationales about the first four spaces, together with an argument space associated with it. This framework is also used to evaluate other existing representations for decision rationales and compare them to DRL in Chapter 7.

considered in similar decisions, the ways in which they were related to other requirements or decisions, the alternatives considered in achieving them, the pro and con arguments, questions generated, and how they were answered. In the following, I illustrate how the PM helps the user to extract relevant information from each type of these rationales. The precedent shown in Fig. 6.5 will be used to illustrate this stage of the algorithm.

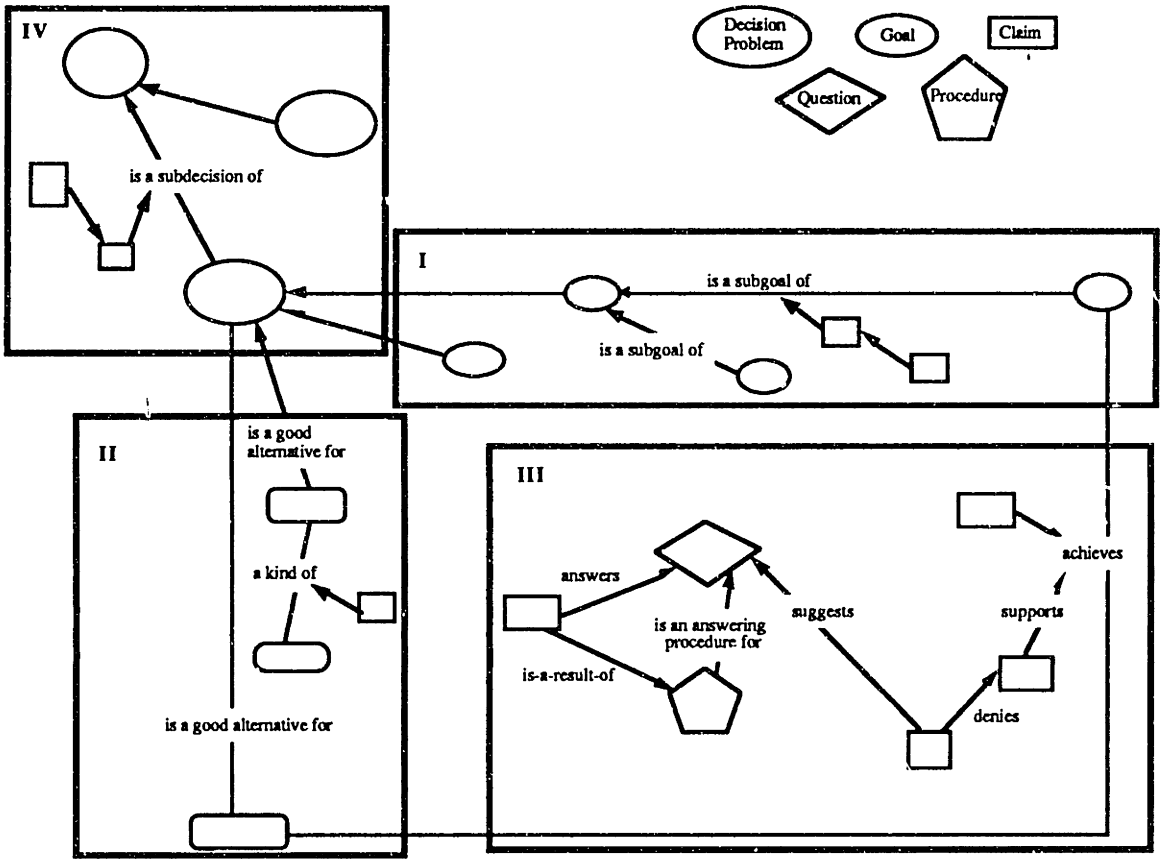


Figure 6.7 Types of reusable rationales in DRL

Type I Rationales: Goals, Their Relations, and Arguments about them

As shown in Fig. 6.8, Type I rationales are captured in DRL by the instances of Goal, their relations (e.g. *Is A Subgoal Of*), and the arguments about them. Starting with the highest level goal matched in the top-ranked decision, the PM presents its subgoals to the user in

order to find out if the subgoals of the matched goal should be the subgoals of the current goal. In our example, the highest level goal matched is the decision problem itself, [Which interface for NEXUS?].

For each of subgoals, SIBYL asks the user whether it should be a subgoal of the matched current goal, i.e. [Which application interface for WING?]. Of the four subgoals shown in Fig. 4.3, two of them, [Highly portable] and [Support WYSIWYG], are already matched to the current goals, namely [Is portable] and [Implement direct manipulation]. Both of these goals are already subgoals of [Which application interface for WING?]. Hence, SIBYL need to only ask about the other two subgoals, [Uniform interface] and [Support over network].

For each of the subgoals presented, the user indicates whether:

(1) it matches one of the existing goals

(2) it does not match any of the existing goals but should be made a subgoal of the matched current goal, or

(3) it does not match and is irrelevant.

The user indicates that [Uniform interface] does not match any of the goals, but is relevant for the current decision. This goal is then installed as a subgoal of [Which application interface for WING?]. Also if either of the two subgoals previously matched, say [Highly portable], was not matched for some reason, the PM would have asked whether it should be a subgoal, and the user would indicate that it should be and that it should also match the appropriate current goal, i.e. [Is portable].

As long as this process yields a subgoal in the first two categories, i.e. as long as there is a subgoal that is to be incorporated into the current decision, the PM recursively performs this process depth-first until there is no more subgoal matched or relevant. In that case, it

pops back to the higher level list, and asks about the next goal in the list. To continue our example, given that [Uniform interface] does not have any subgoal, the PM now asks about the next subgoal, [Support over network]. The user indicates that it is not relevant for the current decision. However, if it made its way into the current decision, the PM would go on and ask whether *its* subgoals, [Efficient communication], should be a subgoal of the current counter-part of its parent goal, [Support over network]. This process terminates after exploring all the subgoals of any goal which matches any of the current goals or has been incorporated into the current decision.

Once it has been decided which of the goals are to be incorporated into the current decision, all the objects (such as claims, counter-claims, questions, and answers) that are linked to those goals and their relations (e.g. *Is A Subgoal Of, Suggests*) are marked as potentially relevant because they tell us something about the goals that have been matched to the current goals.

Type II Rationales: Alternatives, their relations, and arguments about them

Once additional goals have been matched and introduced, the PM helps the user retrieve relevant alternatives. Again, assuming that the PM is proceeding from the top-ranked past decision, all the alternatives in the past decision currently being considered are shown to the user. The underlying assumption is that if the past decision had enough requirements in common to be considered, the alternatives considered for that decision might be potentially relevant for the current decision. In our example, the alternatives considered in the past decision are: [User Interface Management System], [Extensible Interaction Manager], and [NeWs style].

For each of these alternatives, the user can indicate that:

(1) the alternative matches one of the alternatives in the current decision,

(2) the alternative does not match any of the existing alternatives, but is relevant, in which case it is incorporate into the current decision,

(3) the alternative is irrelevant

Continuing with our example, the user examines the three alternatives and decides that [User Interface Management System] matches the current alternative, [IM], at least enough to want to see the rationales about it. Furthermore, although [Extensible Interaction Manager] does not matching any of the current alternatives, the user finds it an intriguing possibility and marks it as relevant.

For each of the alternatives to be incorporated into the current decision (i.e. the case 1 and 2 above), the PM tries to determine whether its sub-alternatives should also be a sub-alternative of the matched current alternative. The algorithm is essentially the same as that used for determining whether subgoals of a matched goal should be a subgoal of the current goal that it matches. After all the alternatives judged to be relevant are determined this way, all the objects linked to the matched alternatives and their relations are marked as relevant because they are arguments, questions, answers, or procedures used to produce the answers about the alternatives matched to current alternatives.

Type III Rationales: Evaluations of alternatives with respect to goals, and arguments about them

Once the goals and the alternatives to be included in the current decision have been determined, the PM brings with them all the objects such as claims, questions, and procedures, that are linked to the Achieves relations between chosen alternatives and

chosen goals. These objects represent the deliberations in the past decision about how well a given alternative achieves a given goal. In Fig. 6.8, the claims shown in bold boxes are all potentially relevant rationales of Type III because they are all linked, directly or indirectly, to the achieves relation between the matched alternatives and the matched goals.

Type IV Rationales: Decision Problems, Their Relations, and Arguments about them

After having extracted potentially relevant parts from a past decision, the PM presents the user with a larger picture of how that decision was related to other decisions. This larger picture would be useful only if the past decision itself, as an instance of Decision Problem, matches the current decision problem, not just when some of its goals match some of the current goals. If they do match, they should appear in the match list for the goals between the two decision problems because Decision Problem is a subtype of Goal (cf. Chapter 3). In our example, the two decision problems, [Which interface for NEXUS?] and [Which application interface for WING?] appear in the match list.

If the past decision problem matches the current decision problem, the PM presents each of the subdecisions of the past decision to the user, and asks if they should be subdecisions of the current decision. Again, the user can indicate that:

(1) the subdecision should match one of the existing subdecisions

(2) the subdecision does not match, but should be made into the subdecision of the current decision

(3) the subdecision does not match and is irrelevant.

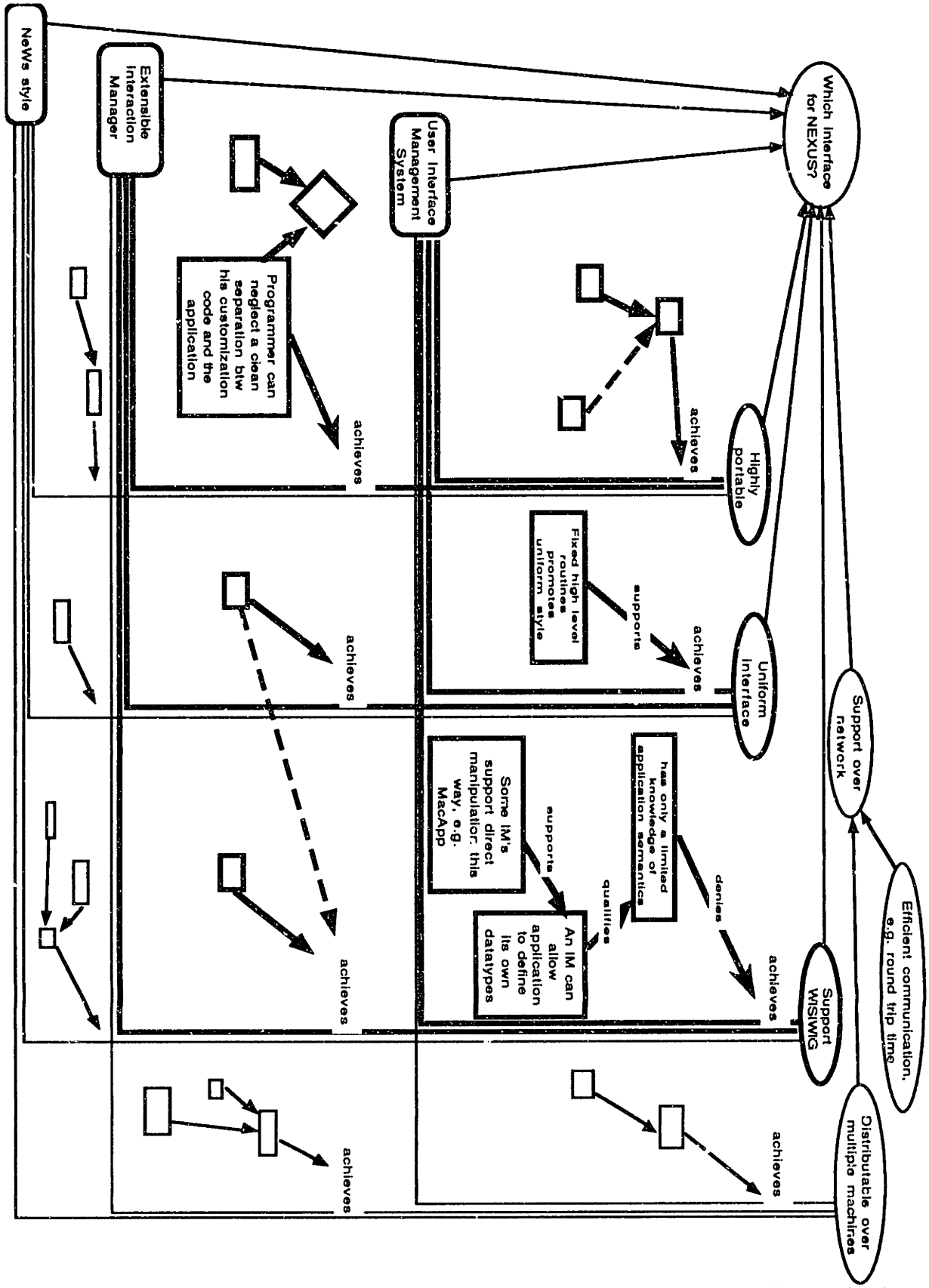


Figure 6.8 The potentially relevant rationales of type III, shown in heavy boxes, in the precedent shown earlier in Fig. 6.5

If the subdecision matches a current subdecision (case 1), the PM records the match so that the PM can later help the user to retrieve useful rationales from the matched subdecision to the current subdecision. In the second case, the PM simply makes the subdecision a subdecision of the current decision. In the third case, the subdecision is ignored. As long as this process yields a subdecision in the first two categories, i.e. as long as there is a subdecision that matches or is to be included as a subdecision of the current decision, the PM recursively performs this process depth-first until there is no more subdecision matched or relevant. In that case, it pops back to the higher level list, and asks about the next decision problem in the list.

Adapting and Linking retrieved structures to the current decision

So far, I have described how the PM helps the user extract potentially relevant rationales from a past decision. These rationales, however, may contain information that are outdated or too context-sensitive that they would not be relevant to the current decision as they are. The PM helps the user to adapt them for the current decision in the following way.

The PM places the rationales judged to be relevant in a new viewpoint. Viewpoints are discussed in Section 6.3. In the present context, a viewpoint can be considered as a working space into which copies of the objects are placed. Hence, copies are made of all the objects in the current decision. The objects that make up the extracted rationales are also copied, and related to the copies of the current decision objects.[16] For example, the

[16] Although this stage is described separately for conceptual simplicity, a fair amount of adapting and linking actually already took place while in the course of the extraction process described above. For example, when the system presents a subgoal of a matched goal and the user confirms that it should be a subgoal of the matched current goal, the subgoal is linked to the current goal. Hence, what is described below is the final adaptation and linking process, not to imply that they do not take place before.

goals from the past decisions that were determined to be the subgoals of a current goal are copied, and these copies are made subgoals of the copy of the current goal. The user then examines this copy of the current decision augmented with the extracted rationales, modifies some of them, rearranges them, or deletes them. For example, the claim, [Mike is familiar with the interaction manager], that supports the claim, [IM achieves the goal Reduce development cost], is deleted because Mike is not a member of the current group. Also, the claim, [Clean separation between wm and application], that supports the claim, [IM achieves Is portable], is made also to support another claim, [IM achieves Common user interface]. After the editing is done, the user installs this viewpoint as the current version, using the version mechanism of Object Lens discussed in Section 6.3. Figure 6.9 shows the final result of incorporating into the current decision the adapted rationales from the past decision.

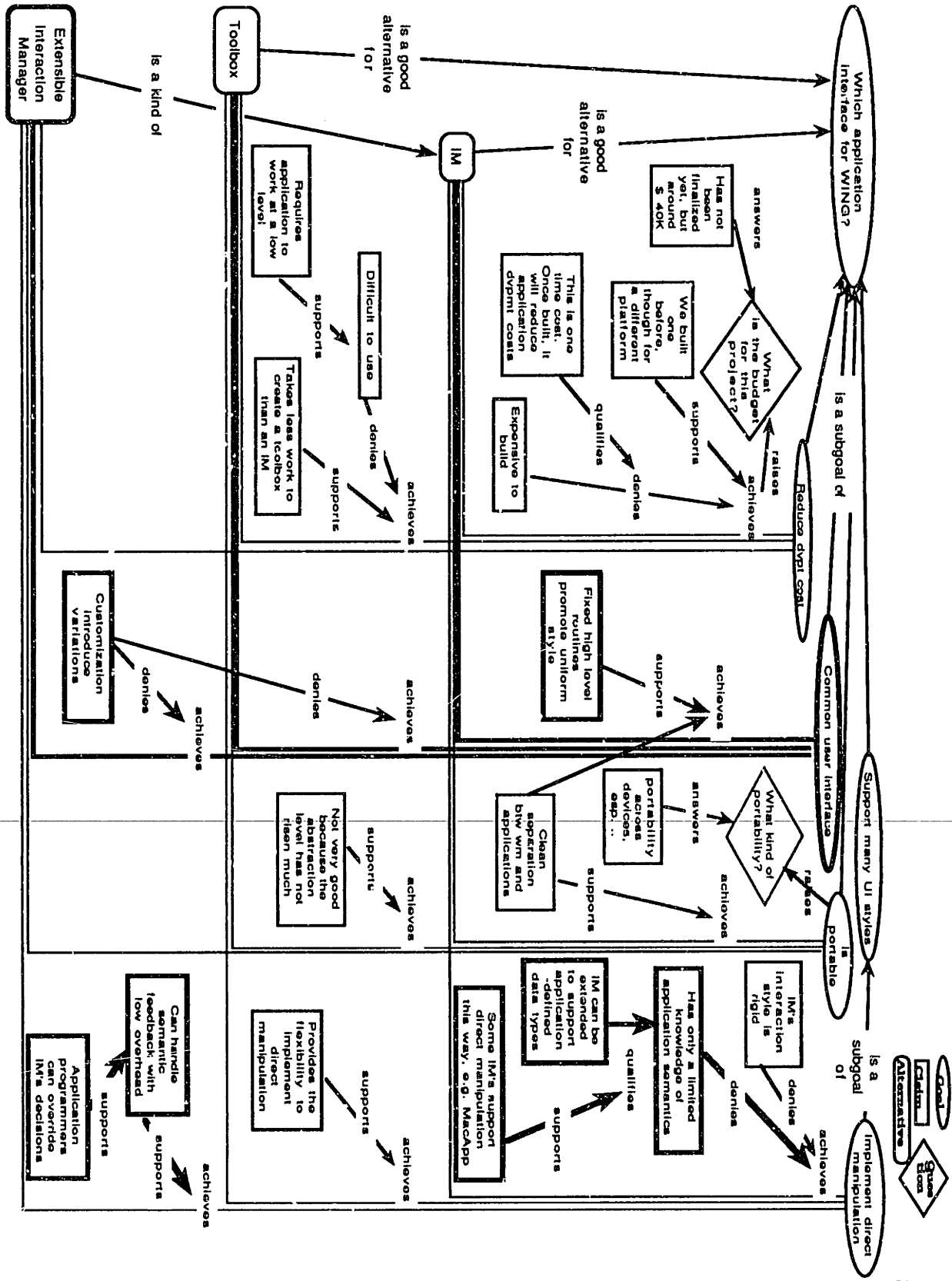


Figure 6.9 The relevant rationales from the past decision, "Which interface for Nexus?", integrated into the current decision, "Which application interface for WING?"

Path:
bloom-picayune.mit.edu!bloom-beacon!gatech!swrinde!mips!spool.mu.edu!cs.umn.edu!aslakson
From: aslakson@cs.umn.edu (Brian Aslakson)
Newsgroups: comp.sys.mac.comm
Subject: Re: Looking for Mac mailservers...
Message-ID: <1991Sep19.182226.24739@cs.umn.edu>
Date: 19 Sep 91 18:22:26 GMT
References: <1CE00001.p05oub@avalon.caladan.wa.com>
Organization: :noitazinagrO
Lines: 33
Cc: minnella@acsu.buffalo.edu

stui@avalon.caladan.wa.com (Stuart Burden) writes:
>In article <...>, minnella@acsu.buffalo.edu (Todd V. Minnella) writes:
> | Hey! My father recently got BITNET access at his college. Does anyone have
> | a list of Macintosh-related mailservers accessible through BITNET?
> | (Ideally, the list would include instructions and addresses.)

Get Eudora get Eudora get Eudora get Eudora get Eudora!!

Try it, you'll like it. You will need a POP3 server, say popper from
berkeley. Unfortunately I'm not sure how to get things via e-mail,
but do this:

"One way to get access the archives is through the BITFTP server
at Princeton. Send a message to bitftp@pucc.bitnet with the body of
the message containing the single word HELP. This should get you
more information, and give you access to any archive site on the
internet." --jwright@cfht.cfht.hawaii.edu from comp.virus

Eudora is available from ux1.cso.uiuc.edu in the directory mac/eudora,
popper is available from ftp.CC.Berkeley.EDU in the pub directory
popper-version.tar.Z, there is also IUPOP3 from Indiana University.
You can get it from the /pub/iupop3/v1.6a directory at logos.ucs.indiana.edu.

popper is a unix implementation of the POP3 protocol, iupop3 is a VMS
implementation of the POP3 protocol based in part on popper.

I think this is what was being asked for???

--
Brian Aslakson
brian@cs.umn.edu (mail)
aslakson@cs.umn.edu (talk)
mac-admin@cs.umn.edu (thru 9/22/91)

6.2 Dependency Management

In decision making, as in other tasks, it is important to maintain the dependencies among the objects being deliberated. The importance of the goal, portability, for example, depends on the outcome of another decision about whether to turn the software into an external product. Or, as in the scenario, whether an alternative should be seriously considered at all depends on how well it achieves the goals which are necessary to achieve. A rationale management system must be able to represent such dependencies in decision making and manage their consequences.

6.2.1 The Scope

A dependency relation in its most general form can be characterized as a pair, (State 1, State 2), which is to mean that when State 1 is true, State 2 should be true. Thus, the dependency relation in our first example above can be characterized as (the decision has been made to make the window manager an external product, the importance of portability is high). The second example is characterized as (the chance of an alternative achieving a necessary goal is low, the evaluation of the alternative is below the threshold). Hence, the most general way of managing dependency is to have the system enforce the relation between the states specified as a pair.

However, this most general form of dependency specification is computationally expensive if the system is not given further information about how to look for a state or how to achieve a state specified. The system would have to constantly watch out for a set of states and would have to be intelligent enough to know how to produce the consistent states. The following describes three ways in which this job is made easier. One is described immediately below. Two others are described by discussing the two versions of the dependency manager in SIBYL.

One way to manage dependency is to restrict the scope of dependency management to a specific kind of dependencies. A Truth Maintenance System [de Kleer 1986; Doyle, 1979;; Lubar 1991] is an example. The job of a TMS is to make sure that when a claim is true, the other claims have truth values that do not produce a contradiction. Thus, the dependency it has to maintain is restricted to that among the truth value attributes of the claims. This type of dependency management, where possible, is clearly desirable for several reasons. The tasks of monitoring a state and producing a consistent state become easier because of the restricted scope. Secondly, a reduced scope often allows us to discover more invariant dependency relations that the system can support. For example, the invariant relation that a TMS supports, i.e. logical consistency, makes sense only among the truth values of claims. Thirdly, such invariant relations, if found, can be hardcoded into the system and can be more efficiently managed. In the present research, there was an attempt to build an evaluation manager, whose job is to maintain the dependencies among the evaluations of claims. Although this attempt produced more research issues than results so far[17], I hope to continue exploring ways to produce dependency managers of restricted scopes.

---

17 SIBYL at one point provided an interface for using existing belief management schemes such as Bayes [Duda et al. 1976] or Dempster-Shafer [Shafer & Logan 1987; Shafer 1976] , to manage dependencies among the evaluations of the claims. However, subsequent attempts to use this interface have revealed many problems that this part of SIBYL has not been seriously used. For example, the assumptions underlying the existing methods -- such as the mutual exclusiveness and exhaustiveness of the hypotheses and the conditional independence of the evidence under a hypothesis -- were often not satisfied in the

Although it is desirable to provide dependency managers for specific kinds of dependencies, we also need a general purpose dependency manager that allows the user to maintain dependency relations that are not instances of the specific kinds. For example, the dependency relation, "if the price of 1 M SIMM goes down below $40, then the importance of the goal "keep the memory requirement below 4 Meg," is too specific to be hardcoded into the system. The user may also want to enforce a dependency relation sometimes but not all the time, e.g. inactivating an alternative that does not satisfy a goal which is necessary to achieve. The dependency manager of SIBYL, described below, helps the user to maintain dependencies in a general way by allowing him to specify explicitly the kinds of dependencies to be maintained and how.

There are currently two versions of dependency manager in SIBYL. One makes use of a slightly extended version of the Object Lens rule system. It provides a high level interface for the user, but is limited in many ways. The other version is more powerful, but does not provide as nice an interface. In the next two subsections, these two versions are described in terms of what kinds of states they allow the user to specify and produce.

---

decisions that we worked with. Another problem was the difficulty of computing the evaluation of an alternative with respect to a goal from its evaluations with respect to the subgoals of the original goal when the subgoals can be related to the parent goal in innumerable ways, e.g. exhaustive, partially overlapping, or mutually exclusive. Eliciting such information from the user seems to require the precision that the user does not have or finds it too expensive to produce. There were other problems such as eliciting an evaluation measure for a claim in the first place, ensuring consistency among the evaluation measures produced by different users, and convincing users to accept automatically computed evaluations. These problems made me focus on the qualitative ways in which SIBYL can help the user assign evaluations, for example by allowing them to make an evaluation in terms of a series of local evaluations through the use of submatrices (cf. Section 7.4), rather than on the autonomous management of evaluations.

6.2.2 Implementation I

One version of dependency management is implemented using a slightly extended version of the Object Lens rule system.

The Object Lens rule system provides the following features [Lai et al.1988].

- agents which can monitor certain types of changes and invoke rules associated with them. The kinds of changes that can trigger an agent include:

  NEW ITEM: a new item appears in the folder specified in its Apply To field, i.e. in a given collection of objects. For example, a new question object is created.

  CHANGED ITEMS:: an item in the folder specified in its Apply-To field changes its attribute, i.e. the evaluation of a claim changes

  AT MIDNIGHT, AT NOON: the clock hits the specified times

  STARTUP, QUITTING: when a session with Object Lens starts or ends

- rules which can be invoked by an agent or by the user.

  The *if*-condition of a rule is specified as a description of an object which can embed other descriptions. Figure 6.10 shows an example. A description stands for all the instances which satisfy the description. Hence, an embedded description allows the user to specify all the instances whose attributes are filled by any instance that in turn satisfy embedded descriptions (e.g. any achieves relation whose associated goal, in turn, has the Importance attribute value, Necessary).

  The *then*-actions that can be executed when the if condition holds are the following.

  MOVE: moves all the objects that satisfy the description in the if condition to a folder,

COPY: copies all the objects satisfying the if description to a folder COPY ITEM: copies all the objects in the specified field of the objects satisfying the description to a folder

CUT: deletes all the objects in the specified field of the objects satisfying the description to a folder

ADD VALUE: adds the specified value to the specified field of the objects satisfying the if description

SEND: sends a message to the specified addresses containing all the objects satisfying the if description

SEND ITEM: sends a message to the specified addresses containing all the objects in the specified field of the objects satisfying the description

Using these features, the user can specify certain kinds of dependencies easily. For example, Fig. 6.10 shows the specification of the dependency that any subgoal of a goal whose importance is low should itself has the importance low. The features of the rule system also make it easier to manage dependencies because both the state to be monitored and the state to be enforced are highly restricted. The state to be monitored is specified in the *triggering* condition of the agent and the *if* part of the rule associated with the agent. The triggering condition specifies *when* to look for a change, the Apply To field specifies *where* to look for the change, and the if-condition of a rule specifies *what* types of change to look for. The state to be enforced is specified by specifying the action that will produce the state.

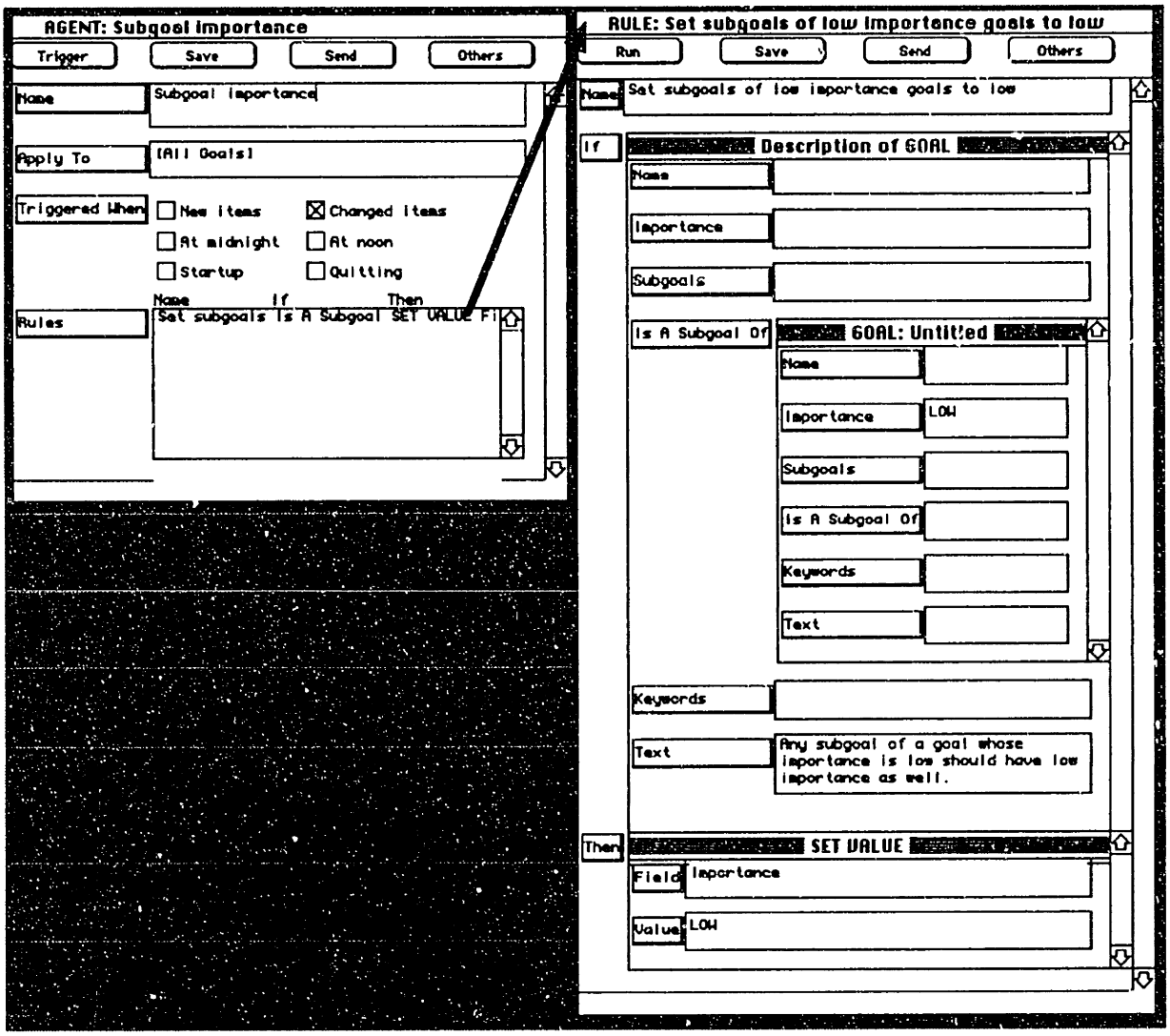


Figure 6.10 The specification of a dependency between the importance of a goal and the importance of its subgoals, which is set to low when the importance of its parent goal is low.

However, because of these restrictions, there are many dependencies that cannot be expressed, including our examples above. The first example, "if the chosen alternative of the decision problem, [Should we make the window manager an external product], is [External Product], then set the importance of the goal [Is portable] to HIGH", is not possible to maintain because the only "then" action that can change an attribute value, ADD VALUE, requires that the object whose attribute value is modified be the same as the object that satisfies the if-description. To overcome this limitation, the Object Lens rule system

has been extended by adding the action, SET VALUE, which allows the specified attribute of the specified object to be set to the specified value. With the SET VALUE action, managing the dependency such as that in our first example becomes possible, as shown in Fig. 6.11.

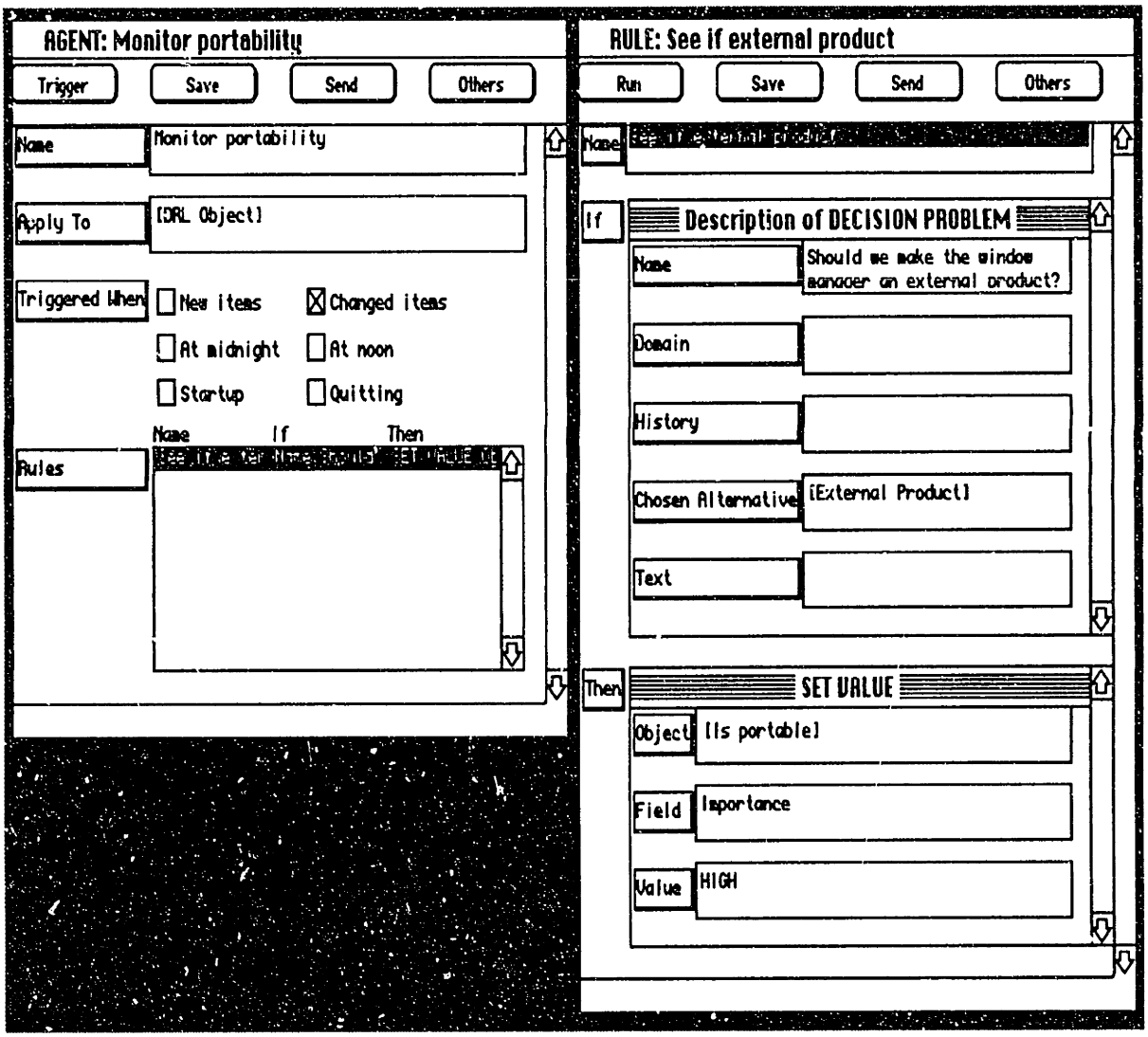


Figure 6.11 The importance of the goal [Is portable] is specified to be HIGH when the window manager is decided to be an external product.

However, this version of dependency manager still has many limitations. In particular, it cannot manage dependencies that involve a state whose specification requires comparison of two objects. For instance, the dependency in our second example cannot be maintained

because in order to say "if any achieves relation whose goal has the importance, Necessary and whose evaluation is Low, then the *Is A Good Alternative For* relation between its alternative and its decision problem should have its Evaluation set to Low", the system needs to map the alternative and the decision problem in the second state to the alternative and the decision problem of the achieves relation in the first state. This version of the dependency manager does not allow us to do that because the rule system does not allow variable binding. Furthermore, it cannot manage dependency whose state involves monitoring a conjunctive presence of values in a given field of an object. Those states that involve disjunctive presence of values in a given field of an object can be expressed, but only awkwardly by creating a separate rule for each disjunctive condition.

6.2.3 Implementation II

In order to overcome these limitations, another dependency manager has been implemented that uses a full-fledged rule system.[18] This rule system allows the user to specify states involving comparison of objects, conjunctive and disjunctive specification of values for a given attribute, as well as change or add attribute values of any object that are bound to variables from the if condition. In particular, Fig. 6.12 shows the way in which the dependency in our second example is represented. Although the states that can be specified in this version of the dependency manager are quite general, the system is helped in managing them by the specification of the types of the objects involved in the if-condition.

Unfortunately, this version of dependency manager achieves these features by allowing the user to use, in effect, any Lisp construct, although some syntactic sugars are provided to

[18] This version of dependency manager is implemented by extending a rule system built by Kum-Yew Lai.

make the specification easier. Thus, to specify these dependencies might require knowledge of Lisp from the user. Also, agents have not been implemented in the version of SIBYL that implements this version of the dependency manager. Therefore, the rules have to be invoked manually by the user. The dependency managers currently implemented in SIBYL are temporary solutions that fulfill the minimum requirement. A future version of dependency manager is planned that integrates the features of the current two versions based on the paradigm of query-by-example [Zloof 1978].

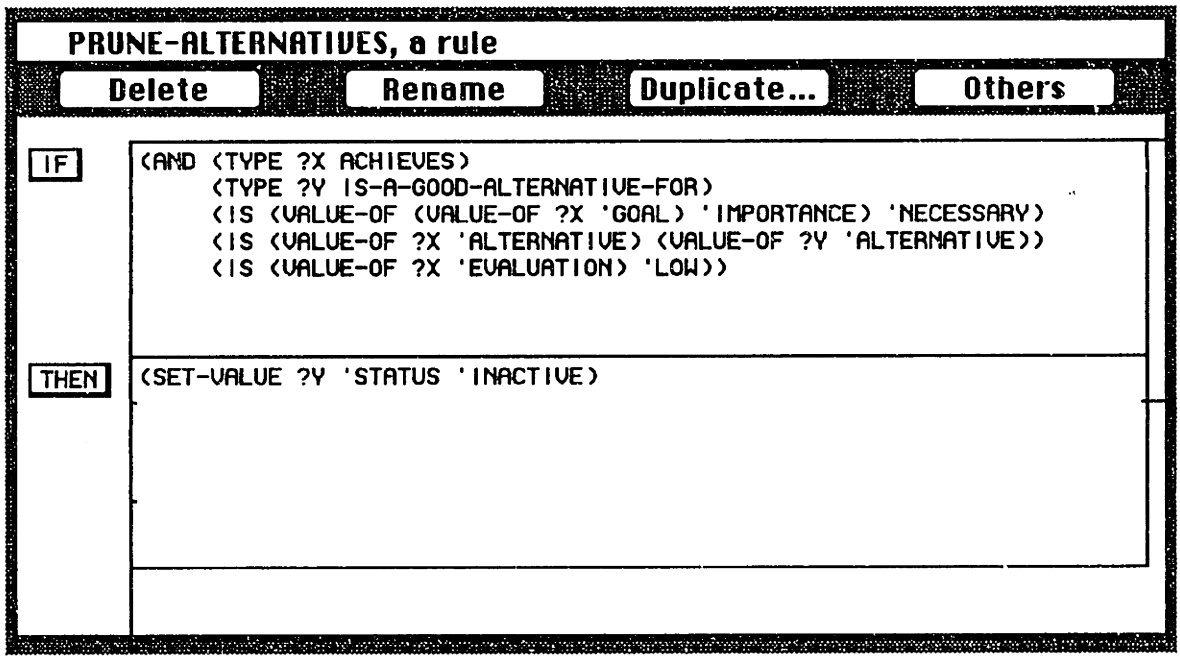


Figure 6.12 Specification of the dependency that if the chance of an alternative achieving a necessary goal is low, the alternative should be inactivated.

6.3 Viewpoint Management

In decision making, it is important to represent multiple decision states and compare them. The following characterize some of the situations that require representing multiple decision states. In the parentheses are example questions that illustrate why representing decision states in that category would be useful.[19]

- decision states that have the same set of objects but with different attribute values. (e.g. What if we reduce the importance of portability? What if we consider this claim more plausible?)

- decision states that have sets of objects that are subset or superset of the others. (e.g. What if we leave out the goal of supporting naive users? What if we include this alternative? What if we did not consider any claim that depended on this fact?)

- decision states that have overlapping sets of objects. (e.g. What if we assume this answer rather than that one? What difference would it make if we assumed this set of subgoals rather than the other?)

---

19 These categories are based on two dimensions along which decision states can differ, one temporal and the other a sort of spatial. The spatial dimension refers to the set-theoretic relation among the objects in the decision states. There are three possible relation between two decision states: the same, superset (or subset), intersecting, and disjoint. The disjoint case is not very interesting. Along the time dimension, a decision state can precede (or follow in time) another. That distinction not very interesting per se, but only when combined with the set-theoretic relation. For example, if a decision state is a superset (or a subset) of another and follows it, the first *elaborates (or simplifies)* the second.

- decision states that are historically related, i.e. ones that have been generated from another in the class. (e.g. What were the decision state last month? Can we revert back to what we had before we introduced this alternative?)

In SIBYL, these multiple decision states are represented as Viewpoints. The Viewpoint Manager in SIBYL helps the user to create, store, retrieve, compare, and merge Viewpoints.

6.3.1 Representation

A Viewpoint is a first class DRL object with an additional attributes, Elements and Viewpoint Relations. The attribute, Elements, points to the objects that are in the viewpoint. Since the goal of a viewpoint is to capture a decision state, it usually points to the decision problem, which in turn points to the objects in it such as goals, alternatives, and arguments.

The field, Viewpoint Relations, points to the relations that link this viewpoint to other viewpoints. So far, I am experimenting with the following relations among viewpoints [20]:

---

[20] When I say "experimenting" with these relations, I mean that these relations are not DRL types yet, but implemented as instances of the general type, *Is Related To*. They are not built into DRL yet, because it is not clear yet whether they are the right set of relations for the task of managing rationales. One way to find out is to experiment at the level of instance.s That is, when we want to represent the relationship *Is A Subset Of* between two viewpoints, we create an instance of the more general built in type, *Is Related To*, and but fills its Keyword field with the keyword *Is A Subset Of*. This way, we can try with different sets of relations easily without prematurely committing ourselves. Also, we can change it more easily because it is easier to change keywords than types. On the other hand, we lose some computational opportunities, like having additional attributes if needed. However, we can still define much of the computational operations to the extent that it is still an instance of the closely related type, *Is Related To* and to the extent that they can be defined on the keyword (e.g. collecting all *Is A Related To* relations whose keyword is *Is A Subset Of* and which is related to the current viewpoint.)

Is the next version of (historical)

Is a subset of (subset)

Has different weights on goals (same structure with different attributes)

Has different evaluations on claims (same structure with different attributes)

Has alternative assumptions (overlapping structure)

Since the goal of viewpoints is to represent and compare multiple decision states that often share objects, it is important to make clear in what sense objects are shared among viewpoints. On the one hand, objects shared by different viewpoints (e.g. the goal, portability), need to preserve their identity across the viewpoints. Otherwise, viewpoints cannot be compared because they are about different objects. On the other hand, an object shared by multiple viewpoints should be able to have different attributes in different viewpoints. Otherwise, viewpoints would not be able to capture situations of the first kind above (e.g. different weights on the importance of portability) nor the last kind (i.e. historical versions). Furthermore, given that in situations of the third kind (e.g. argue under an assumption), a viewpoint is used to confine the changes to an object to the local context, an object shared by multiple viewpoints needs to have different manifestations. Hence, in SIBYL, an object shared by multiple viewpoints is represented as a copy in each of these viewpoints, but all the copies of a shared object share the same identifier. The implementation of this representation is discussed in the next subsection.

Because viewpoints are first-class objects, we can relate them and browse through them. Figure 6.13 shows several viewpoints and their relationship. Furthermore, viewpoints can appear as alternatives in a meta-decision problem. Figure 6.14 shows a decision problem, "Which weights on goals?" In this example, viewpoints 1 and 2 capture the same decision state except that the importance assigned to the goals are different. The goals for this decision include meta-level concerns such as "Err on the safer side" and "Consistent with

our corporate policy" that may be important in deciding how important the object-level goals (e.g. "Support window manager over a network", "Implement direct manipulation") should be.

6.3.2 User Interface

When the user wants to create a new viewpoint, he creates a new instance of Viewpoint, as usual by double-clicking on the type in the type window, and inserts a link to it in the Viewpoint attribute of the decision problem. The user can also relate the new viewpoint to other viewpoints by inserting in its Viewpoint Relation attribute links to other instances of the *Is Related To* relation. For example, if the user wants to indicate that the new viewpoint has different weight assignments on goals from the one in the viewpoint that the decision problem was previously associated with, then he inserts in its Viewpoint Relation attribute, a link to an instance of *Is Related To* which relates the two viewpoints and has its Keyword attribute with *Has different weights on goals*.

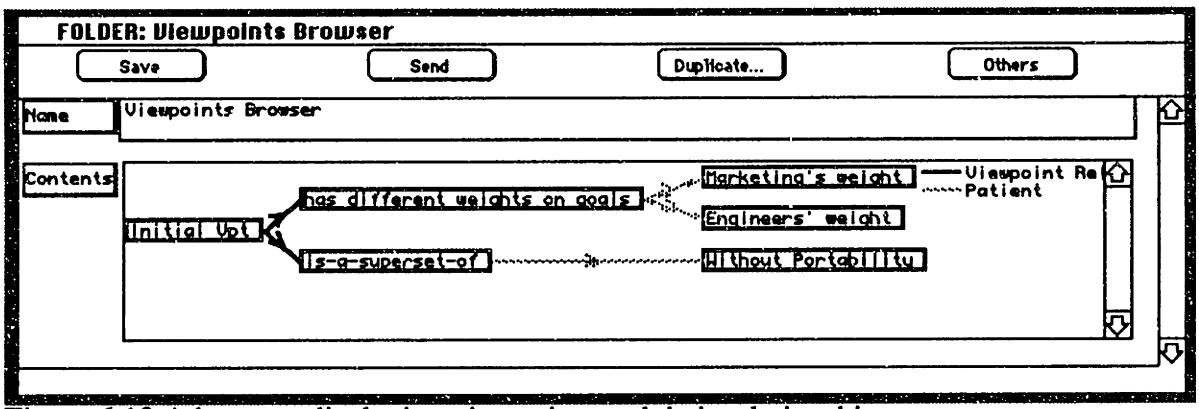


Figure 6.13 A browser displaying viewpoints and their relationship

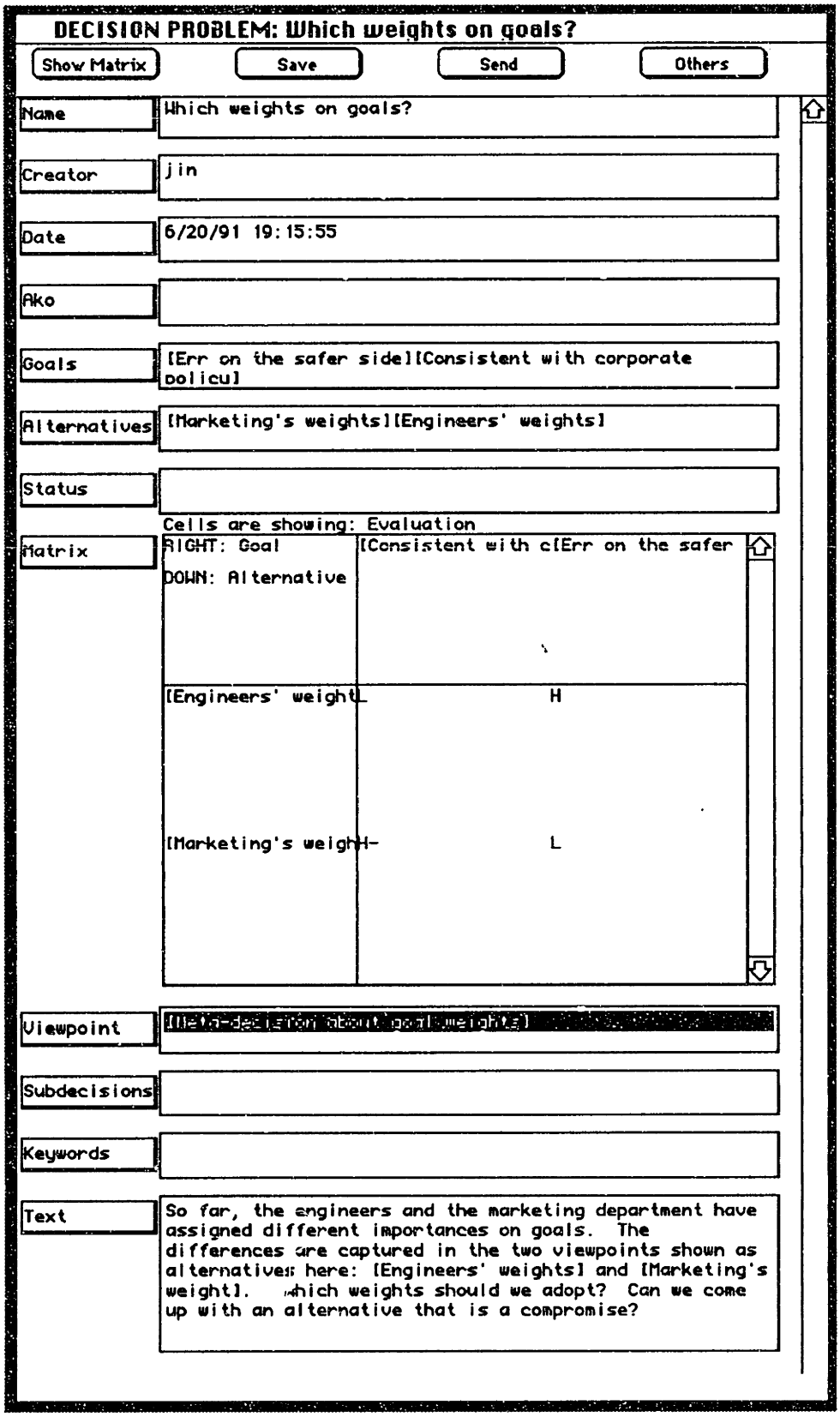


Figure 6.14 A meta-decision problem about which of the viewpoints, [Marketing's weight] and [Engineers' weight], to adopt as weights on the goals.

With a link to the new viewpoint in the Viewpoint attribute of the decision problem, the user executes the action, Save Viewpoint, from the menu. The viewpoint manager then makes a copy of the decision problem as well as of all the objects that it contains while preserving their id's. This new copy of the decision problem would be the same as the current one except the new viewpoint that its Viewpoint field points to . Any change or addition made to this viewpoint would not be visible to other viewpoints, and the objects that belong to different viewpoints, including the decision problems themselves, can be examined side-by-side, as well as saved and loaded independently.

I will illustrate this process by going through how the example in the scenario is actually implemented. Suppose that the user wants to consider the decision problem without portability. First, if he has made any change in the current viewpoint since the last time that it was saved, he saves it again. Then he deletes the goal, portability, from the current decision, which will automatically delete the corresponding column in the decision matrix as well. Then he creates a new instance of viewpoint, specifies its name to be "Without portability", and inserts a link to it in the Viewpoint relation of the decision problem. Then he chooses the Save Viewpoint action to save the now current viewpoint. Figure 6.15 show the viewpoint and the relation created, and the two decision problems under the old and the new viewpoints. Now the decision problems in the two viewpoints can be compared, added to, modified, and saved independently of each other.

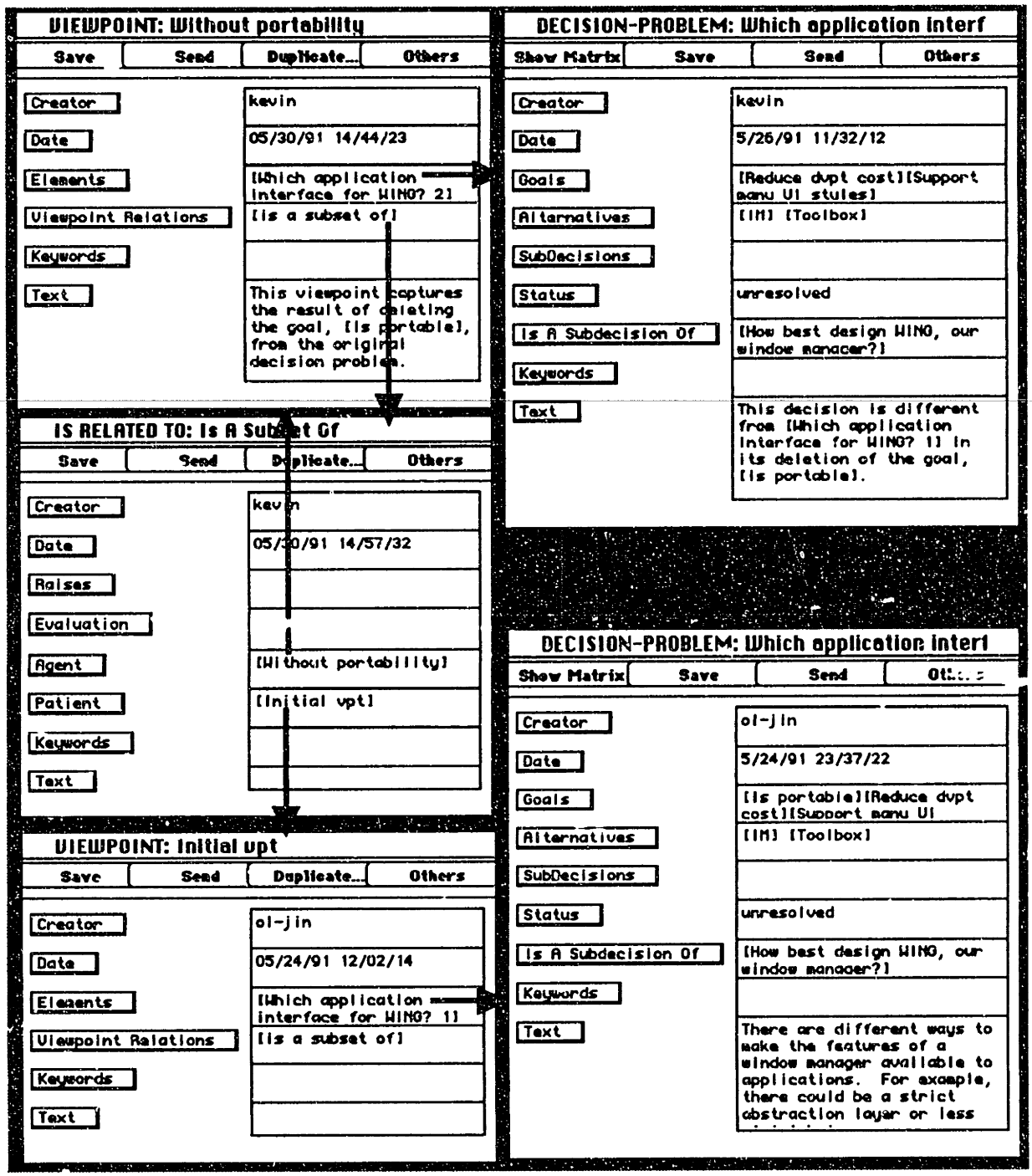


Figure 6.15 Two viewpoints, one of which is a subset of the other. The first viewpoint, [Without portability], contains the decision problem that is derived from the original decision problem by ignoring the goal, [Is portable]. The latter decision problem belongs to the viewpoint, [Initial vpt], of which the viewpoint, [Without portability] is a subset.

6.3.3 Implementation

As discussed above, an object shared by viewpoints needs to be both the same and yet different in some sense. This is done by using two identifiers, object id and version id, for an object, as consistent with the Object Lens mechanism.

In Object Lens, associated with an object are two identifiers: object id and version id. The object id is created when the object is created, it is assigned a unique id that is a concatenation of a time stamp and a machine name. The object id does not change throughout the lifetime of the object. At the same time, it also gets a version id which is a time stamp. The version id changes every time the user modifies the object and is saved or sent to other users.

This version mechanism is extended so that when a new viewpoint is created, all the objects in it, i.e. those pointed to in its Element field, keep the object id's as they have in the current viewpoint. When the viewpoint is saved using Save Viewpoint, however, all the objects get new version id's whether they have changed or not. Hence, an object shared by multiple viewpoints has the same object id but different version id's. This way, an object is able to maintain its identity across viewpoints through its object id while it can be changed locally within a viewpoint without affecting its counterparts in other viewpoints.

6.4 Other Services

In this section, I describe the services of SIBYL that are not captured by the three categories described in the previous sections. The services described below, except the last (Summary and Explanations at various levels), use the various features of Object Lens unmodified, some of which have been discussed in the previous two chapters. The purpose of the following discussion is to categorize and make salient the services that these features are put to use in the specific context of SIBYL.

Bookkeeping

SIBYL helps the user to keep track of the rationales across sessions (Problem Category II[21]) by retrieving relevant objects of interest. Using the Object Lens rule system described earlier, SIBYL can show the user things such as:

- all decisions that are yet to be resolved
- all decisions that the current decision depends on

---

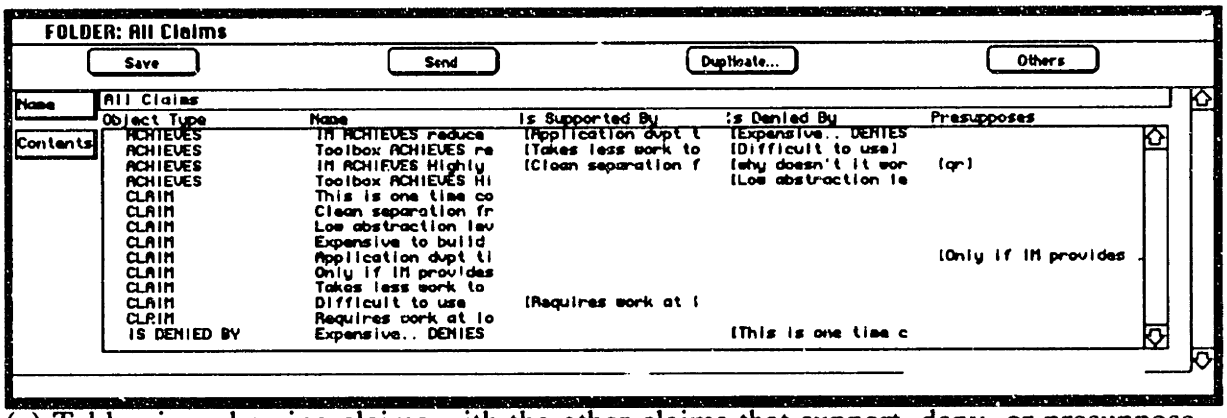
21 These Problem Categories refer to the taxonomy of the problems discussed in Chapter 1 that arise in group decision making: reusing rationales across decisions (Type I), managing rationales across sessions (Type II), sharing rationales across groups (Type III), and qualitative support within a session (Type IV). The Precedent Management address the Type I problems. The section below on sharing objects through electronic mail (6.4.3) and the Viewpoint Management (6.3) address the problems of Type III. The dependency and the viewpoint management address the problems of Type IV.

- all the arguments that have been created since the last time the user examined the decision
- all the claims that support this claim
- all the objects that have been created by a specific person
- all the questions that have not been answered
- all the goals whose importance is HIGH

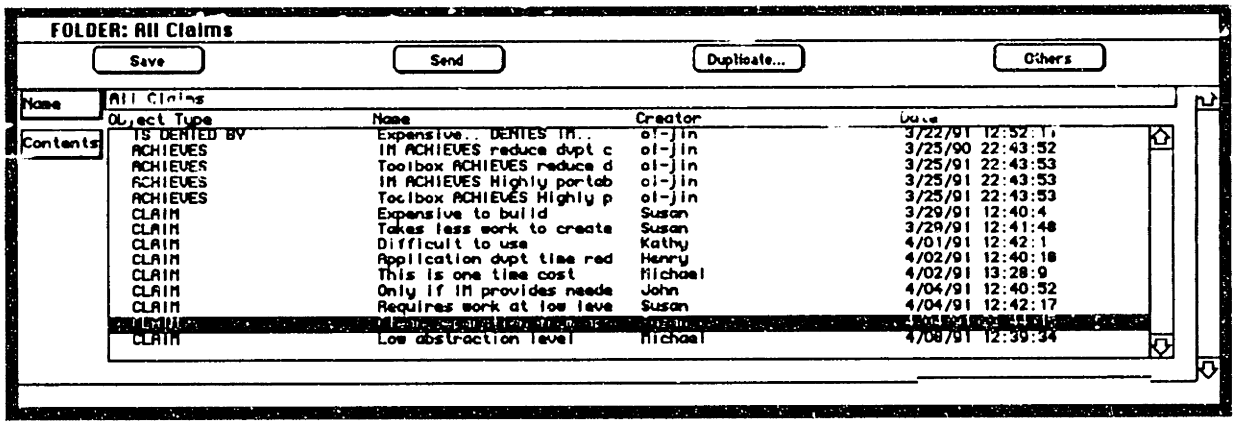
Furthermore, the user can examine them in different ways, using the display formats provided by Object Lens. The user can see them as a table, as a network, as a matrix, or as a calendar (if the objects include the Date field). Figure 6.16 shows a set of claims in these display formats. The user can specify which of the attributes should be shown in each of these displays. For example, instead of or in addition to showing the claims that each claim supports and denies as shown in Fig. 6.16a, the table could also display the creator and the modification dates of the claims. Also, in the network display, the user can specify any other relation to be shown, instead of or in addition to the *Supports* and *Denies* relations among the claims as shown in Fig. 6.16b.

Monitoring and Notification

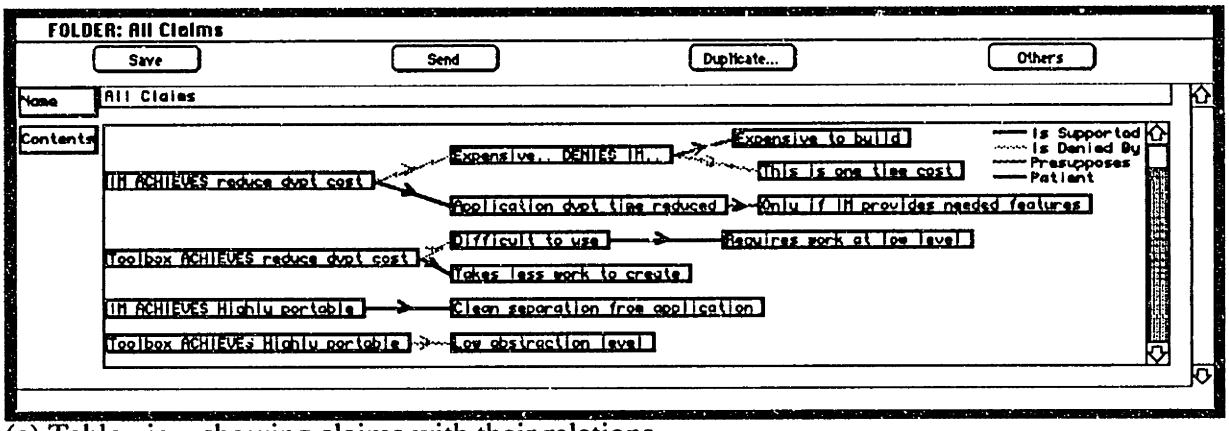
As described in the section on dependency management, the actions of the rules that an agent can trigger include Send. This feature can be used to automatically send a message to the users satisfying a specified description when a change of certain type occurs. For example, the members of the current decision making process can be notified when any decision that the current decision depends on changes or when an urgent question is raised. Figure 6.17 shows a rule that illustrates our first example.



(a) Table view showing claims with the other claims that support, deny, or presuppose them



(b) Table view showing claims with their creators and modification dates



(c) Table view showing claims with their relations

Figure 6.16 Different display formats displaying all the claims in a given decision problem

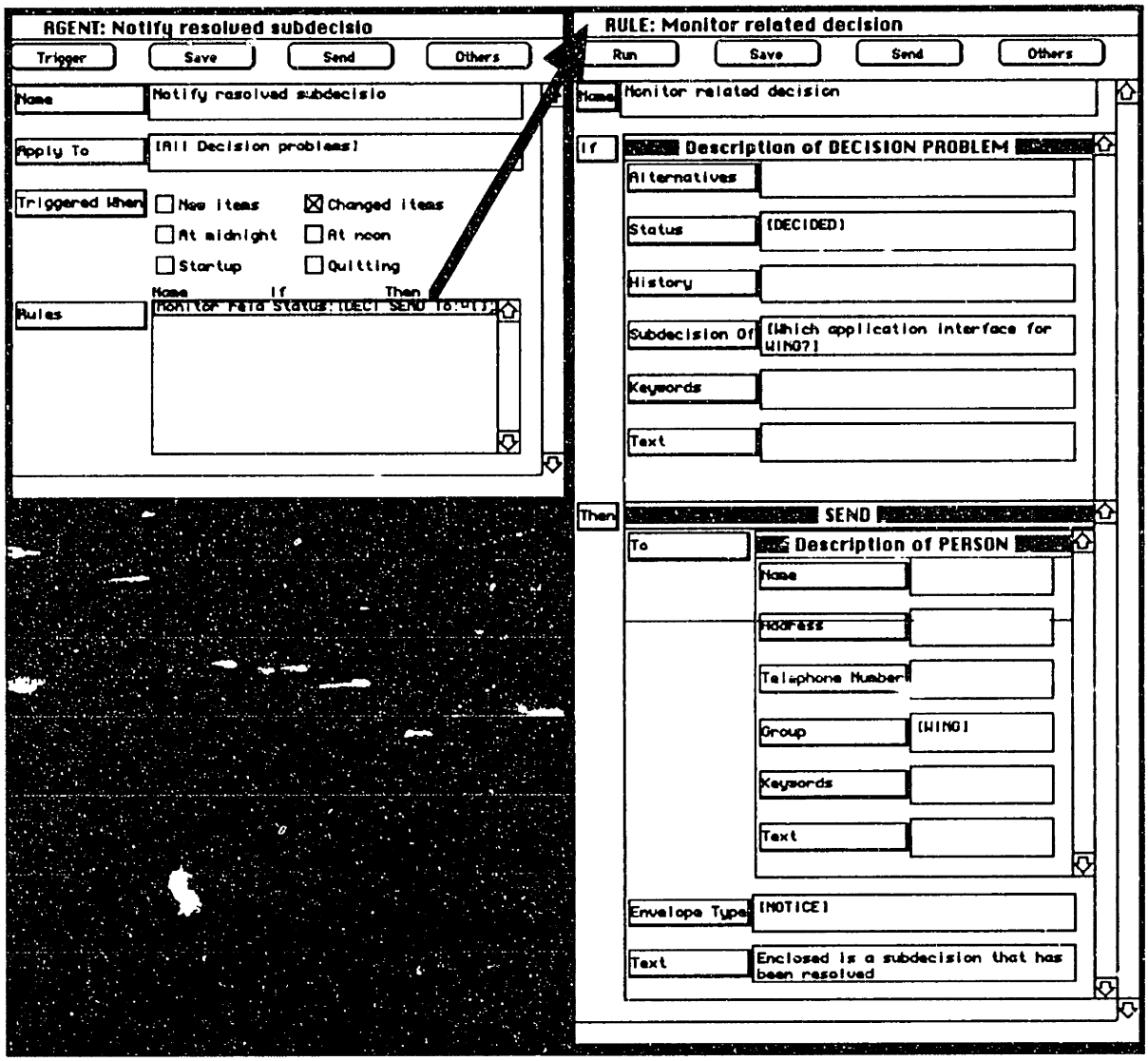


Figure 6.17 An agent with a rule that monitors the statuses of subdecisions and notifies a person when they are decided

Sharing Objects through Electronic Mail

The users of SIBYL can send any collection of objects through electronic mail. As described in Chapter 5, the users of SIBYL can use this feature to participate in the decision making process through electronic mail by sending SIBYL objects, such as goals and claims, to others. The user can also use this feature to send relevant rationales to other

people who are not participating in the decision. To illustrate with an actual example, after SIBYL has been used to discuss how to implement a feature in Object Lens (e.g. whether a particular command should appear in the field menu or in the window menu), somebody who was not aware of this discussion raised the same issue. At that point, the person was sent the rationales that captured this discussion. Similarly, this feature of sending objects can be used to support sharing rationales across different groups (Problem Category III).

Summary and Explanations at Variable Levels

SIBYL provides a summary of the current evaluations of the alternatives at various levels using its features such as decision matrix, submatrices, and argument browsers.

The topmost level summary, i.e. the overall ranking of the alternatives, is shown in the order in which the alternatives appear in a decision matrix. This ranking is determined by the evaluation measures of the *Is A Good Alternative For* relations associated with the alternatives. The alternatives whose evaluation measures have not been assigned yet appear after those which have been fully evaluated, thus conveying the information about the ranking among the alternatives. The next level summary is the evaluation of the alternatives with respect to each of the goals at the top level, i.e. immediate subgoals of the decision problem. This summary is shown by the evaluations in the cells of the decision matrix, which represents how well a given alternative achieves the goals shown at the top of the matrix.

The evaluations at this level of summary are explained by presenting the user with the rationales underlying each of these evaluations, which is shown by either a subgoal matrix or an argument browser. If the user wants to know why an alternative has the current

evaluation with respect to a given goal, SIBYL displays the submatrix associated with the goal. A submatrix shows the evaluation of the alternatives with respect to the subgoals of the goal if any (Fig. 6.18). The evaluations in a submatrix explain the evaluation of an alternative with respect to a goal in terms of its evaluations with respect to the subgoals.

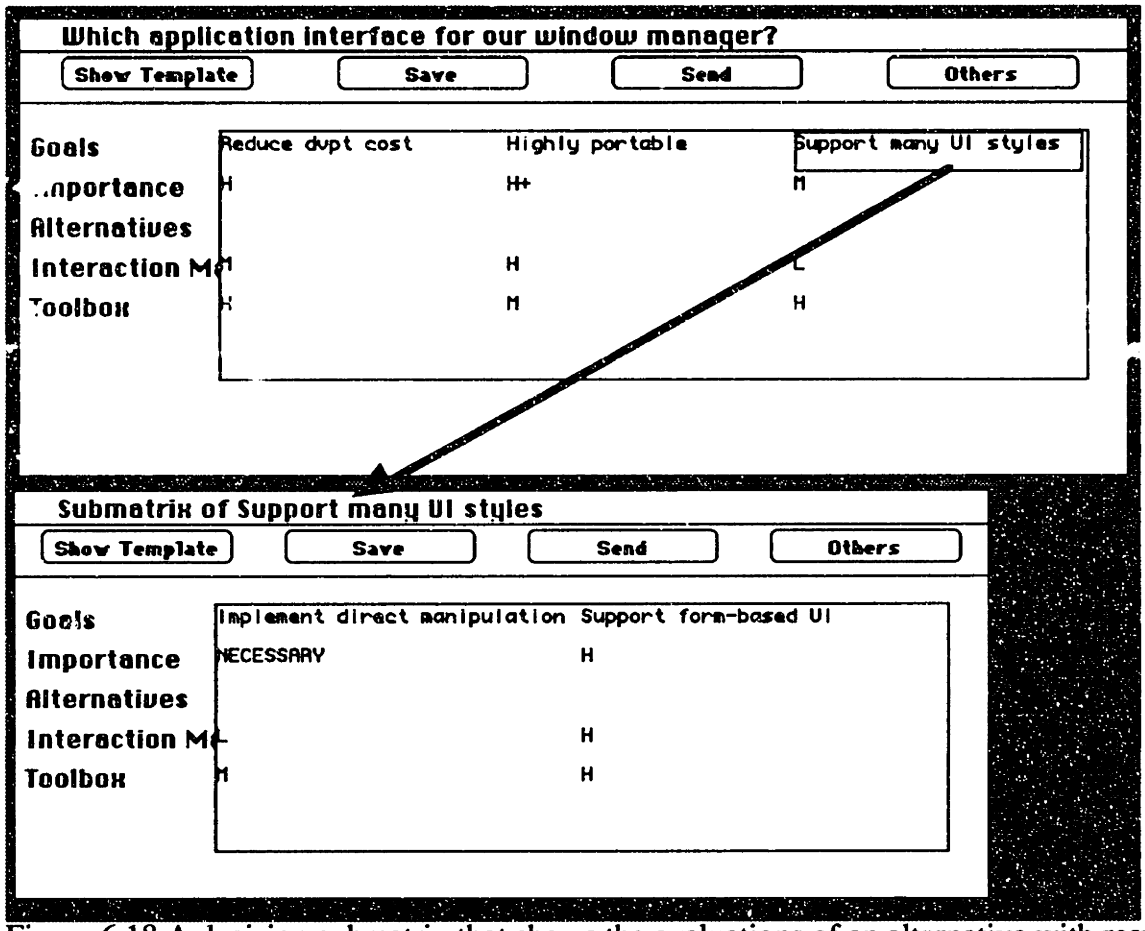


Figure 6.18 A decision submatrix that shows the evaluations of an alternative with respect to the subgoals for a higher level goal

The user can also ask SIBYL to display the argument browser associated with a given evaluation in a decision matrix (Fig. 6.19). An argument browser shows all the claims, questions, or answers, that have been cumulated and used to arrive at the current evaluation. If the subgoals specify their parent goal exhaustively, that is if achieving all of the subgoals is exactly equivalent to achieving the parent goal, then the evaluation of an alternative with respect to the parent goal would be entirely specified in terms of the

evaluations with respect to its subgoals. In that case, only those goals which have no subgoals would have argument browsers associated with them. However, in general, a goal is not specified exhaustively by its subgoals. For example, the goal [Support many UI styles] is not achieved fully by achieving its subgoal [Implement direct manipulation]. In that case, the argument browser associated with a cell in a matrix displays those arguments relevant in evaluating the alternative with respect to the goal which are not covered by its subgoals. This way, SIBYL provides summaries of evaluations and explanations of the summary at multiple levels of details

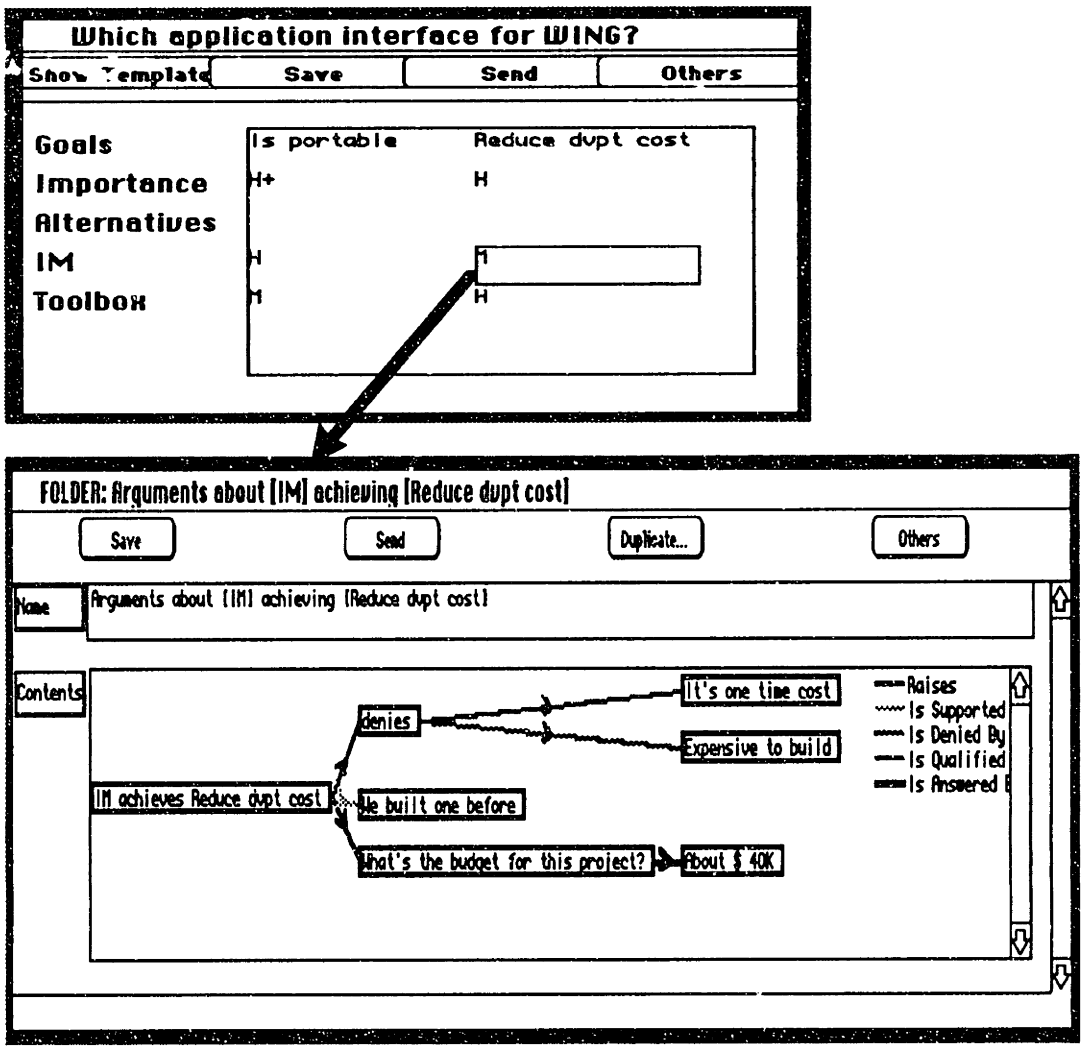


Figure 6.19 An argument browser that displays the arguments responsible for the evaluation of an alternative with respect to a given goal.

Chapter 7

Comparison to Related Work

Studies abound on decision making, ranging over multiple discipline such as management [DeSanctis & Gallupe 1987; Huber 1984; Nunamaker et al. 1988; Sol 1987], computer science [Duda 1976; Kraemer & King 1988; Pearl 1988], and psychology [Kahneman et al. 1982; Kleimuntz 1990; Pitz & Sachs 1984; von Winterfeldt & Edwards 1987]. Most of these studies are relevant to the research on rationale management system. They all tell us about some aspects of decision making which could be incorporated to improve the design of a rational management system: a better representation, a better environment for capturing the rationales, and better decision support.

While acknowledging the vast amount of potential research on decision making, the comparison here will be restricted to the research on *tools* that support decision making.

The body of related research is further restricted to the tools that capture both the problem solving process and the rationales about the problem solving process. This restriction leaves out, for example, tools such as expert systems that do not keep a trace of their reasoning, but leaves in those expert systems that can explain their decisions [Davis & Lenat 1982; Swartout 1986].

7.1 Systems that Capture Rationales

The tools in this category represent explicitly the process that they went through in making a decision or in solving a problem in general. Presumably the rationales are captured to provide some service, and have to be managed. Therefore, the tools in this category are rationale management systems in the widest sense.

There are many dimensions along which these RMS's can be compared. For the purpose of contrasting the present work with others, two dimensions are useful: the extent to which an RMS requires formalization of the domain knowledge and the extent to which the rationales it manages can be reused. By domain knowledge, I mean the knowledge of the subject about which the decision is made. For example, if the decision is about a window manager design, then the domain knowledge includes the knowledge about the objects (window managers, applications, the components of a window manager such as graphics toolkit), their properties, and their relations. Some RMS's require that this knowledge be completely formalized, others are based on completely informal knowledge (i.e. natural language text), and yet others allow mixture of formal and informal knowledge. The difference in this dimension results in a difference in the range of tasks an RMS can support and the services it is able to provide. RMS's can also differ in the extent to which the rationales they manage can be reused.

Figure 7.1 shows four major categories of RMS's placed with respect to two dimensions; each of them is discussed in more detail below.

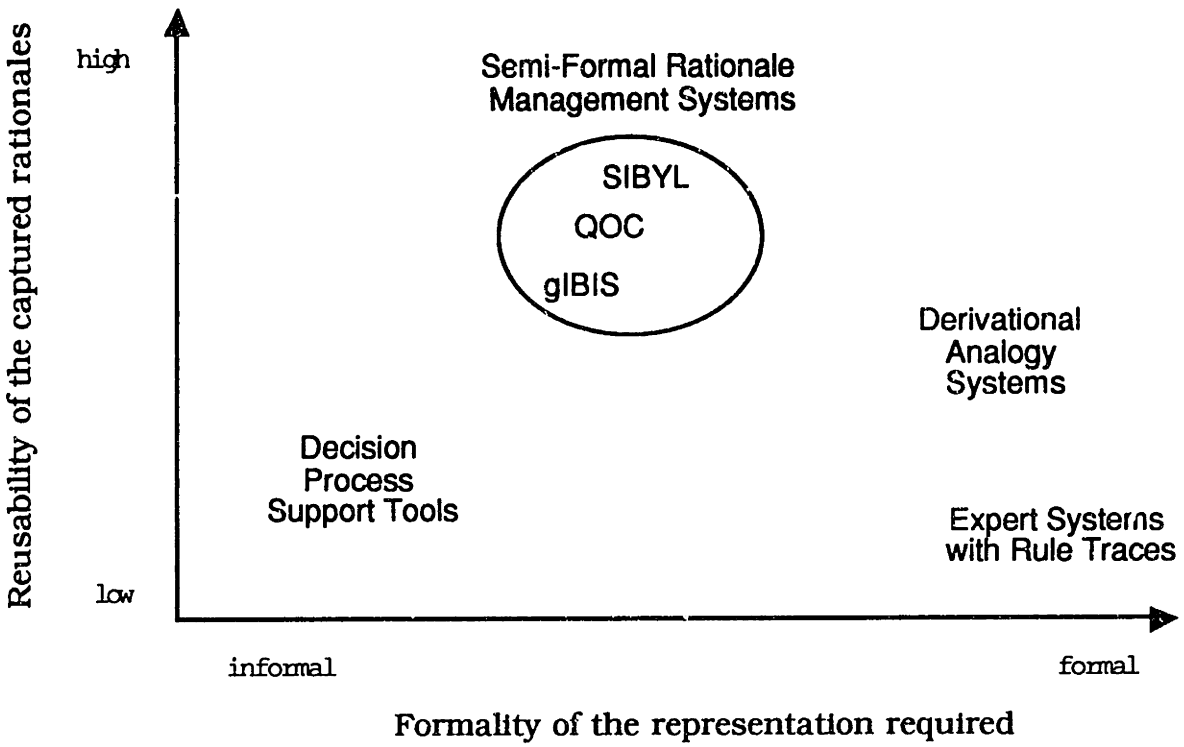


Figure 7.1 Classification of existing rationale management systems along two dimensions: formality of the representation required and reusability of the captured rationales

The first category includes systems that are usually labelled as group decision support systems [DeSanctis et al. 1987 ; Kraemer & King 1988; Nunamaker 1991]. The primary goal of these systems is to support and structure the *process* of group decision making. Some of these systems, for example, provide better communication and presentation support for group decision making, such as tele-conferencing facilities, large shared displays, vote tabulation and display. These systems often record the sessions of their use, either in form of audio or video output or natural language text. Other systems help users

to structure their ideas by providing support for brainstorming or allowing them to use modeling techniques such as decision analysis or social judgement analysis. In these systems, the representation of the rationales is more structured, such as in decision trees. Although in either case, the recorded rationales can be reused if we are willing to spend a lot of time and effort identifying the relevant parts of the rationales, the cost is often too prohibitive to be justified by the resulting benefit. Nor is reusing rationales a goal of these systems.

Then there are systems that capture rationales in completely formal language. Expert systems that keep traces of their rule invocation are examples. The domain knowledge is captured by these rules and the objects that are referred to in their if and then conditions. The traces of these rules are rationales in the sense that they embody the reasons for the decisions that the systems made. In fact, these traces are used to answer questions that the user might ask, such as Why a certain action was taken or How an action was implemented [Davis & Lenat 1982]. However, again, reusing these rationales is not a goal of these systems, and their reusability is quite limited because reuse requires understanding of the goals and the plans, not just a sequence of actions.

The third category includes systems that have been labelled as derivational analogy systems [Huhns & Acosta 1988; Mostow 1989; Steinberg & Mitchell 1985]. Their primary goal is to reuse the rationales captured in the process of solving problems such as design or implementation of artifacts. Hence, these systems explicitly address the issues that arise in reusing rationales such as adapting them to new context, and provides techniques for resolving them. Although these systems allow the user to interactively guide the reuse process, the rationales are generated and captured by the system in formal representations.

The reusability of the rationales in these systems are limited in several ways. First, the rationales that they capture are limited primarily to the goal space. That is, the rationales are usually captured in the form of plans executed or rules fired, and are indexed by the goals that they achieve. They are reused when the goals that they achieve arise in the new context. Few systems capture the rationales about the alternative, the evaluation, and the issue spaces, that is the different alternatives that they considered in solving their problems, the evaluation measures that they used to choose an alternative among them, and the relation between this decision and other decisions. That is not to say that these systems do not consider different alternatives; in fact, rule-based systems use various conflict resolution strategies to choose among the alternatives. Also, the backtracking that these systems often do in the course of deciding which plans to apply constitute the issue space aspects of rationales. However, these rationales are not captured explicitly, and therefore cannot be reused. Finally, the rationales about the argument space is missing altogether because these systems do not produce evaluations of the alternatives by arguing, but by simply following an algorithm that is fixed and built into them.[22]

Another way in which the rationales of the derivational analogy programs are limited is that the tasks that they support are fairly narrow in scope. First of all, these systems, or more generally their approach, would not be applicable to domains whose knowledge cannot be currently formalized. Even in domains where such formalized knowledge exists, such as generation of grammar-driven tools (POPART [Wile 1983]) and a circuit redesign (REDESIGN [Steinberg & Mitchell 1985], BOGART [Mostow & Barley 1987], ARGO [Huhns & Acosta 1988]), the representations and the techniques used limit the use of these systems to fairly narrow subset of these tasks. For example, ARGO, which is designed as

---

22 These limited rationales are not inherent in these systems, but are true of the current systems. We can certainly imagine a system in this category that captures and reuse the rationales of the other kinds. The reason why the existing systems do not points to the second reason why their reusability is limited, namely the degree of formalization that they require.

a "domain-independent system for applying analogical reasoning to the design process", is applicable to domains where strictly top-down refinement is possible. Also, in general, automatic replay and reuse of rationales have yet to solve many problems such as missing preconditions, the reference problem, localization, and context-sensitivity [Mostow 1986]. Furthermore, as Mostow notes in his survey of these systems, "Conspicuous deficiencies [of these systems] include insensitivity to higher-level aspects of redesign problems [such as environmental criteria or the cost consideration], and the lack of a retrieval method that scales up efficiently to larger designs and design libraries." [Mostow 1989, p.172-173]

This limited reusability of the rationales produced by these systems stems from their goal of providing as much automated support as possible, and the problems that they raise and address are important ones relative to that goal. The fact nevertheless remains that there are many problems for which managing rationales is important and that this approach taken in the derivational analogy systems is not currently applicable to this task.

Another approach to exploring automated support is through incremental formalization, namely to start with a base system that is practically useful and then gradually automate parts of it as we in fact use the system. The next category, which is labelled as semi-formal rationale management systems and includes SIBYL, groups the tools that take this approach. These systems use semi-formal representations, which allows informal descriptions to coexist with or within the formal constructs that they provide. By letting some parts of the rationales to be described informally and interpreted by human users, these systems gain in the scope of the tasks that they can support and avoid the brittleness in their support. As a result, the rationales they capture are general and highly reusable. Since the tools in this category are most closely related to SIBYL in their goal and representations, they are discussed separately in the next subsection.

7.2 Semi-Formal Rationale Management Systems

The systems described in this section are most similar to SIBYL in their goals and representations. They all aspire to be practically useful and capture rationales in semi-formal representations. However, at the next level of detail, they are different in each of the three components of a rationale management systems: representations, methods proposed for using them, and computational services that are defined on the captured rationales. These differences, in turn, lead to the differences in the reusability of the rationales that they manage. The following discussion presents these systems and then compares them to SIBYL.[23] In particular, their representations are discussed and compared in terms of the models developed in Chapter 3. Doing so makes clear what components they decided not to represent explicitly and what the resulting consequences are. The discussion is quite detailed not only because their comparison to SIBYL requires elaborating their details, but also because the discussion illustrates the rationales underlying the design of DRL.

IBIS (Issue Based Information System)

IBIS was developed in [Kunz & Rittel 1970] to represent designers' argumentation activities. One IBIS variation is that used by gIBIS [Conklin & Begeman 1988], "a hypertext tool for exploratory policy discussion." Because gIBIS is most well-known and

[23] An expanded version of the following discussion, which includes some other related representations, like Toulmin's [Toulmin 1958], appears in [Lee & Lai 1991].

has demonstrated "industrial strength" [Yakemovic & Conklin 1990], we discuss gIBIS first. Other variations of IBIS include PHI (Procedural Hierarchy of Issues) [McCall 1987] and the one used by [Potts & Bruns 1988] for the rationale module in their representation. They are discussed briefly following the discussion of gIBIS.

gIBIS (Graphical Issue Based Information System)

Figure 7.2 shows the objects and relations that form the language of gIBIS. Figure 7.3 shows an example representation, in which someone raises an *Issue* such as where to put the window commands. *Positions* are created that *Responds-to* the issue. *Arguments* are created to *Support* or *Object-to* a Position. An *Issue* can be related to other objects.

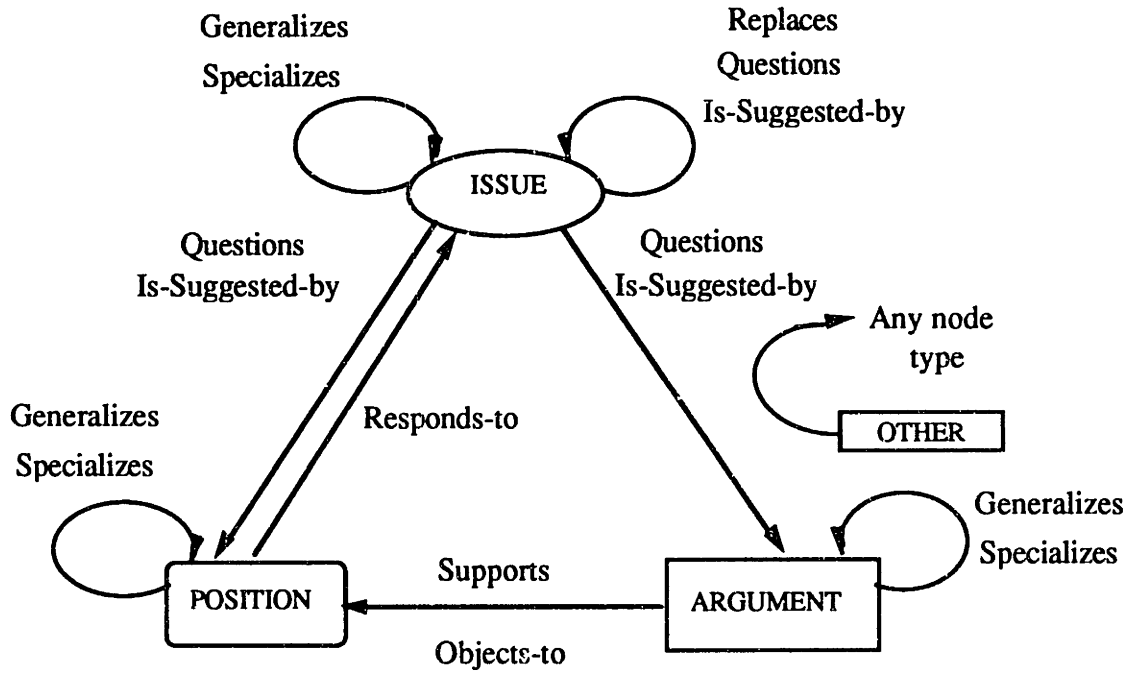


Figure 7.2 The vocabulary of gIBIS

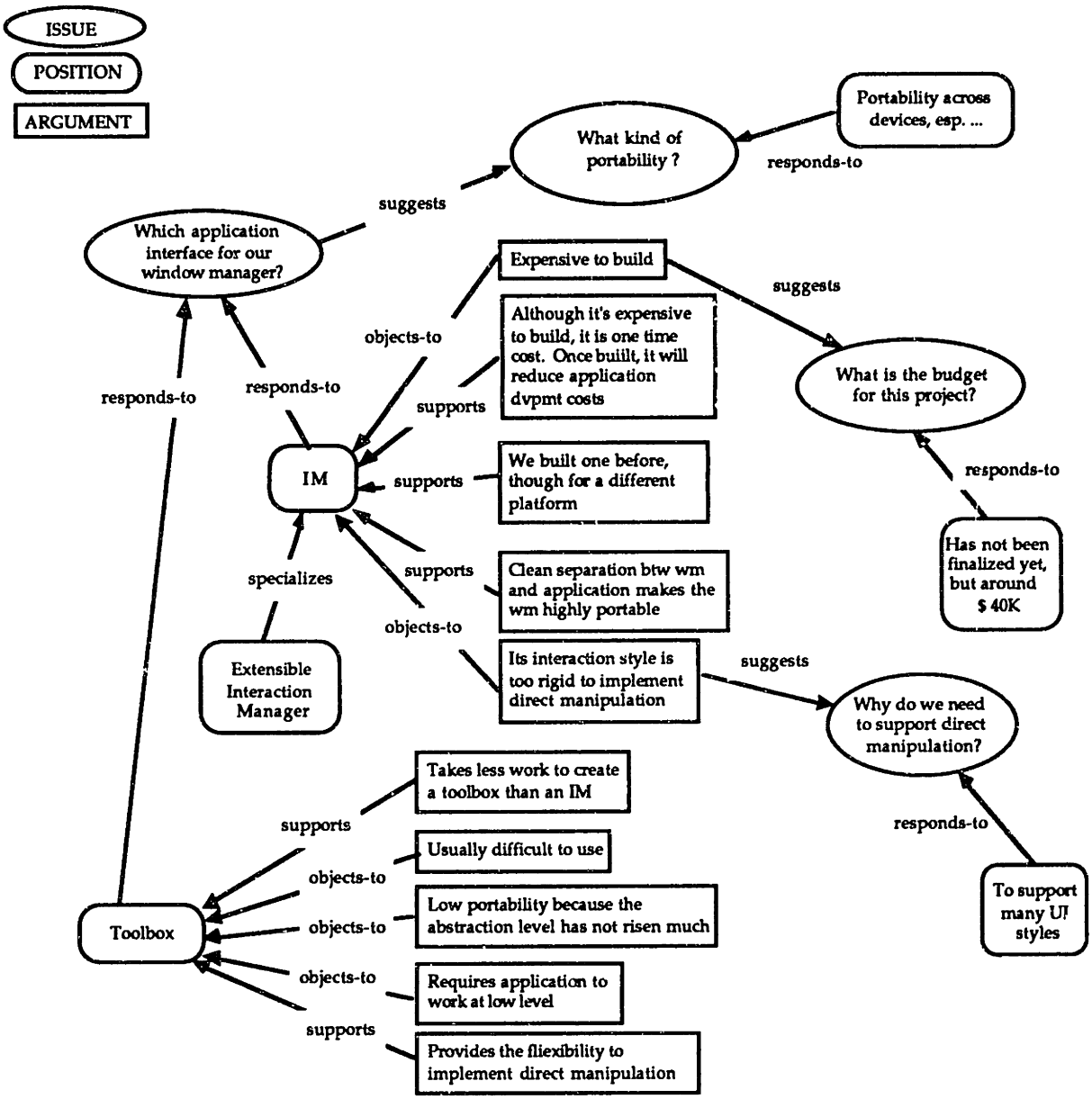


Figure 7.3 An example rationale represented in gIBIS

The scope of the gIBIS representation depends on what an issue is. If we take an issue in a very general sense to mean any question that takes a set of positions, then the *Issue/Position/Argument* structure can represent a fairly large part of the design rationale spaces. The internal structure of these spaces, however, is not well differentiated in gIBIS.

The *alternative space* is represented in gIBIS by *Positions* and the relation among them. Since multiple *Positions* can be created for a given issue, gIBIS allows the representation of multiple alternatives, thus offering at least the richness of our model 2. The only relation, however, among the *Positions* that we can represent in gIBIS is the *Specializes/Generalizes* relation, although there are other relations that can connect a *Position* to objects of other types (e.g. *Questions* or *Is-Suggested-By*).

This relatively poor expressiveness in the argument space in gIBIS has several consequences.[24] First, you cannot qualify an argument. For example, we cannot indicate that an *Argument* A is valid only if another *Argument* B is valid. Furthermore, since relations are not claims, as in DRL, there is no way of saying that we agree with A and B, but not that A *Supports* B. This is because *Supports* is not explicitly represented as an object in gIBIS and is not something that we can argue about. Being able to argue about relational claims is important. One may agree with a claim but not that the claim denies another claim. For example, the user may agree that the interaction manager is expensive to build but not that this claim denies the claim that interaction manager reduces the development cost because he believes that the reduction in the application development cost will more than compensate for the initial cost.

There are also some things that gIBIS can say but only awkwardly. For example, in order to express the relationship that one *Argument* supports another *Argument*, we have to create an *Issue* that *Is Suggested By* the *Argument* to be supported. This issue is about

---

[24] Whenever we say that a representation cannot express some information or has limited expressiveness, we do not mean that people cannot infer that information from the representation. For example, we keep a detailed enough record in natural language of what happened, or even a video recording of the whole design process, we can always retrieve the information that has ever been expressed if we work hard enough. When we say that a representation cannot express some information, we mean that the representation does not provide constructs that make the information explicit in such a way that help people easily see the structure or make it amenable to computational manipulation.

whether the *Argument* is right. We next create appropriate *Positions* (e.g. "Yes" or "No") that respond to the new issue, and then argue about these *Positions*. This way of representing argument relations may lead to proliferation of objects. We might be able to reduce the proliferation with an interface that hides the intermediate details. However, a more serious limitation of this way of representing, as opposed to allowing claims to be directly responded as in DRL, is that we cannot answer questions such as "Show me all the arguments that respond to this argument." We might try to answer the question by following the *Is Suggested By* link that connects the original argument and the many issues that might be responding arguments. However, the *Is Suggested By* link is too general for this purpose because it does not distinguish the issues that contain responding arguments from that do not.

The *criteria space* is beyond the scope of gIBIS. This is a serious limitation for a design rationale representation language. Since criteria are not explicit, we cannot argue about them; we cannot represent the reasons for having these criteria; nor can we indicate any relationship, such as mutual exclusiveness, among the criteria. Further, when criteria change, there is no easy way to accommodate the changes. It is more difficult to isolate the real disagreements among people because the criteria they use in their arguments remain implicit. The reusability of the rationales is also reduced because goals are important bases for judging relevance, as discussed in the section on the precedent manager. The explicit representation of goals can also provide modular representations of arguments (6.4) and multiple viewpoints (6.3).

The *evaluation space* used by gIBIS consists of some nominal categories such as "Rejected" and "Chosen" assigned to the *Positions*. We could use finer categories, such as "Waiting for More Information" to give more detailed information about the status of Positions, Arguments, or Issues. However, a more sophisticated evaluation management

might require information that gIBIS is not able to provide because of its limited expressiveness in the other spaces, such as how the criteria or the arguments are related, how important each criteria is, and whether a claim is denied or only qualified.

The unit of the *issue space* in gIBIS is an *Issue*, and gIBIS provides several constructs for describing the relations about issues. An *Issue* can *Generalize*, *Specialize*, *Replace*, *Question*, and *Suggest* another *Issue*. In particular, as an issue or a decision often gets reformulated and differentiated, the relations such as *Replace* and *Specialize* seem essential. It would be nice, however, if we can somehow show whether a given set of relations is complete or adequate.

DRL and gIBIS have similar structures at least at a high level. *Issue* in gIBIS corresponds to *Decision Problem* in DRL, *Position* to *Alternative*, and *Argument* to *Claim*. However, as we have discussed above, gIBIS is limited in expressiveness. Therefore, DRL can be viewed as extending gIBIS in several ways: an explicit representation of the criteria space, a richer representation of the argument space, and the provision of an infra-structure for defining evaluation measures. The gIBIS structure has the advantage of being simple, and it is an empirical question what this simplicity buys us. The foremost criteria for a language is not whether it is simple, but whether it is helps users accomplish their tasks. An expressive language can be made easy to use with an appropriate user interface, but it is impossible to make a usable language more expressive. Therefore, it seems that a good starting point is to design an expressive language that is capable of providing more useful services rather than one that is simple to use.

The *method* of using the gIBIS representation is simple. All the nodes and relations that have been created, such as shown in Fig 7.3, are displayed as a network in a window. The user augments this network by examining this network, adding new nodes, and linking

them to the existing nodes. The system provides many features that make it easier for the user to do so, such as a zoom window that shows a small part of the network in detail, an IPA node that aggregates all the nodes about a given issue into a single node, a pop-up menu on each node that informs the user of the legal actions available on that node. Since this process of augmentation resembles using the representation with a paper and a pencil, but much easier, gIBIS is easy to learn. In SIBYL, on the other hand, the mapping between the user interface (e.g. decision matrix, argument browser) and the underlying representation is not as straightforward. As a result, SIBYL has more initial learning cost.

The *computational services* in gIBIS are primarily hypertext features that help the user navigate through the network, such as those mentioned above. On the other hand, a few systems have been built that provide interesting computational services using gIBIS. In particular, [Lubar 1991] has implemented a truth-maintenance system that manages logical dependencies among the gIBIS nodes. fmIBIS [Greene et al. 1991] has extended gIBIS so that informal nodes can be linked to the formal specifications that they give rise to. These formal specifications can be checked by a theorem prover, and their results can, in turn, be used as arguments in the gIBIS network for supporting or objecting-to a position. These systems show that a rationale representation language like IBIS can produce interesting computational services with limited expressiveness, and the tradeoff between expressiveness and computational services offered will need to be explored in the future.

Other IBIS-based Representations

We have seen how gIBIS measures against our models. We now measure against other IBIS-based languages. PHI (Procedural Hierarchy of Issues) [McCall 1987] overcomes some of the gIBIS limitations by allowing a quasi-hierarchical structure among issues,

answers and arguments. The semantics of the hierarchical relation is different for the different spaces. In the issue space, if *Issue* A is a child of *Issue* B, that means A "serves" B — that is, resolving A helps resolving B. In the answer space (i.e. the alternative space), an *Answer* is a child of another if the first is a more specific version of the second. In the argument space, an *Argument* is a child of another if the first is a response to the second. Hence, unlike in gIBIS, we can respond to an *Argument* directly by making it a child node of the *Argument*. Furthermore, PHI is *quasi*-hierarchical and allows sharing nodes (i.e. multiple parents) and cyclic structures. While the quasi-hierarchy increases the expressiveness of PHI, the system is still limited by the shortage of relations. An *Issue* cannot be specialize another *Issue*, an *Answer* cannot server another *Answer*, etc. Therefore, the same comments about not explicitly representing relations in gIBIS apply to PHI.

There are many parallels between PHI and DRL. The quasi-hierarchical relation among *Answers* in PHI corresponds to the *Is a Kind of* relation in the alternative space in DRL. The quasi-hierarchical relation among *Issues* in PHI corresponds to the *Is a Subdecision of* relation. The quasi-hierarchical relation among *Arguments* in PHI is specialized into the *Supports* and *Denies* relations in DRL. But many constructs in DRL find no correspondence in PHI, such as *Goals*, *Presupposes*, or *Is a Subgoal of*, and *Is a Part of*. Thus, DRL can be viewed as pushing further the extensions that PHI made to IBIS by generalizing the hierarchical structure to more complex relations and making explicit some other elements, especially those in the criteria space.

JANUS [Fischer et al. 1989] is interesting as an attempt to bridge two representations. One of its components, CRACK, uses a rule-based language for representing domain specific knowledge, e.g. about kitchen design. The other component, ViewPoints, uses PHI to represent the rationale for the decisions they make. JANUS integrates the two

representations by finding the appropriate rationales represented in PHI for the particular issue that designers face in the construction phase while using CRACK. Although the current interface is limited to that of locating the relevant parts of the representations, bridging a design rationale representation and a domain representation is a very important topic of research because such a bridge can allow us to represent the relations among the alternatives or the criteria in more domain specific ways.

[Potts & Bruns 1988] outlines a generic model for representing design deliberations, shown in Fig.7.4. The model extends the IBIS model to represent the derivation history of an artifact design. One starts with an abstract *Artifact*, such as a plan for a formatter, associates with it the *Issues* that arise in making the plan more concrete, associates with each of the *Issues* the *Alternatives* considered, some of which lead to a more concrete plan, and so on until we make the plan concrete enough to be implemented. Associated with each *Alternative* may be a *Justification* The internal structure of rationale, as shown in Fig.7.4, is essentially an IBIS structure, and much of what we said about IBIS applies to this representation as well. However, by describing a series of progressively more concrete *Artifacts*, and *Justifications* for its path, this model represents the alternative space and its argument space better.

To the extent that the representations discussed in this section -- PHI, JANUS, and the Potts and Bruns system -- rely on IBIS, they inherit several limitations from it. There are no constructs for representing the criteria used for evaluations; an argument cannot directly respond to another; and we cannot argue about relational claims. In the latter two cases, however, the component which uses IBIS or its variant is modular enough that DRL can be viewed as an alternate representation for the component. In Potts and Bruns', DRL can be used for representing the rationale component. In JANUS, DRL can be used as the alternative representation for ViewPoints, i.e. the issue-base module. We believe that

DRL can provide a better interface in JANUS with its other component, CRACK, because DRL is more expressive and can better support knowledge-base operations.

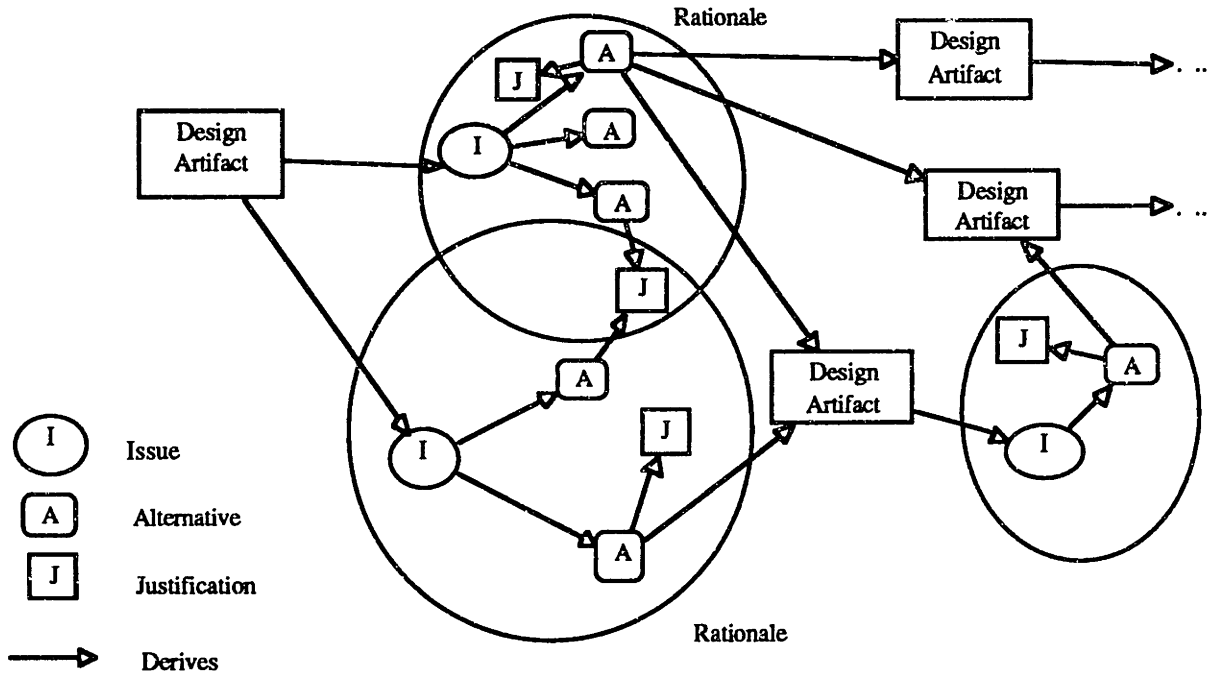


Figure 7.4 The schema for the Potts and Bruns representation of design rationales

QOC (Question, Option, and Criteria)

QOC is a representation proposed by [MacLean et al. in press] for "constructing" design rationales. A design rationale in QOC is said to be not a record of the design process, but instead is a co-product of design that has to be *constructed* alongside the artifact itself.

The major constructs of QOC are straightforward and map clearly to the framework proposed in this paper. Figure 7.5 shows an example represented in QOC.[25] The unit of the *issue space* in QOC is a *Question*. The unit of the *alternative space* is an *Option*. *Questions* and *Options* roughly correspond to *Issues* and *Positions* in gIBIS. However, unlike gIBIS and like DRL, QOC can represent the *criteria space* with *Criteria*. A *Criteria* (e.g. "Implement direct manipulation") is said to be a "bridging criteria" if it is a more specific one that derives its justification from a more general one (e.g. "Support many UI styles"). The units of the *evaluation space* are links labelled with "+" and "-",

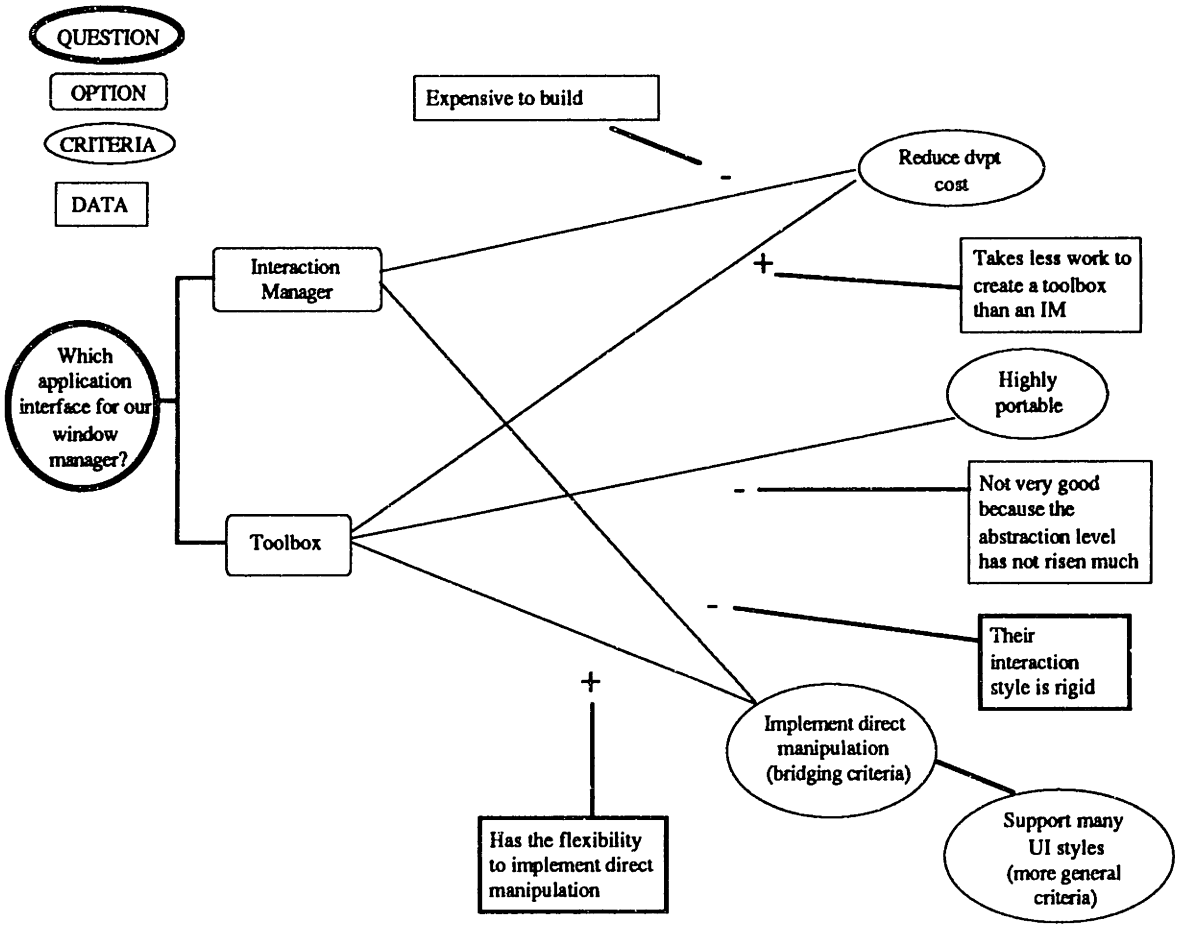


Figure 7.5 An example rationale represented in QOC

25 We could represent only a part of our example because it was not clear to us how to represent the rest in QOC.

corresponding to whether an option does or does not achieve a given criterion. The constructs for representing the *argument space* are *Data, Theory,* and *Mini-Theory.* One supports the evaluation ("+" or "-") of an option with respect to a criterion by appealing to empirical *Data* (e.g. "Expensive to build") or to an accepted *Theory* . When there is no relevant data at hand or existing theory to draw on, the designers may have to construct a *Mini-Theory,* which is an approximate explanation of part of the domain.

QOC as we understand it has several limitations as a representation language. First, in the *argument space,* constructs like *Data, Theory,* or *Mini-Theory* do not seem to capture many aspects of arguments. For example, a claim like "IM can be extended to support application-defined data types." seems neither a datum nor a theory. Nor is it clear whether and how we can argue about theories, or individual claims in theories. In the *alternative space,* there is a reference to cross-option dependency, but no specific constructs are discussed for representing it. In the *criteria space, Bridging Criterion* is described as if it is a special type of *Criterion.* If so, that is building a role into a fixed type, which results in unnecessary inflexibility. That is, being a *Bridging Criterion* is not a property inherent in the object itself but in the relation that the object has to another criterion. As such, one should not have to classify a given object as a Bridging Criterion but only as a Criterion while indicating that it is a bridging criterion for another object through the relation it has with the latter. Otherwise, we have to unnecessarily change object types depending on which object we focus on.

The QOC constructs for the evaluation space are "+" and "–" links. These evaluations are said to be supported by appealing to empirical data and accepted theories. It is not clear, however, how we can accommodate changes in evaluation status due to incremental support. Suppose that the evaluation of the *Option,* "Interaction Manager," with respect to the *Criterion* "Reduce development cost" is initially "–" because of the empirical *Data*

"Expensive to build." Further suppose that we then realize that there is another argument (which we again represent using *Data*) in favor of the *Option* with respect to the same *Criterion* (e.g. "Application development time is reduced"). We can at this point augment the design rationale in one of two ways: (1) create an additional "+" evaluation link between the same *Option/Criterion* pair, or (2) keep the original link but change its "-" to reflect the net effect of the two *Data* (one pro and the other con). The latter option has the disadvantage of losing information about which *Data* was appealed to support the net evaluation since the link between any *Data* and the evaluation link is not labeled (such as with "Supports" or "Objects to"). The first option of keeping two evaluation links avoids this problem, but we are not sure whether QOC allows multiple evaluation links from one *Criterion* to one *Option.*

QOC is perhaps the closest to DRL at least in its basic structure. Both have the five spaces clearly delineated, although the constructs for the argument space are less clear. *Decision Problem* maps to *Question*, *Alternative* to *Option*, and *Goal* to *Criterion*. *Claims* map roughly to *Data*, *Theory*, or *Mini-Theory*, depending on whether the claims are empirical statements or parts of an established theory, or an informal theory. The concept of subgoals maps to the concept of *Bridging Criteria.* Also, at least some of the links can be argued about: e.g. the evaluation link between an option and a criterion can be supported by *Data* or *Theory.*

The *method of using QOC* has been articulated more than any of the systems discussed so far. This method is specified partially in the form of the properties that QOC constructs should have. For example, for something to be a Criterion, it "must be unconditional in the sense that, other things being equal, the greater the extent to which the Criterion is met, the better is the design"; it "must be evaluative, i.e. it must be a single-valued measure of some property of the design, with a definite sense of higher values being better" [MacLean et al.

in press]. The method is also specified in the form of "heuristics" that tells the user how to use QOC profitably: for example, "Use Options to Generate Questions," "Consider Extreme Options," "Identify Options which Generate Dependencies," and "Look for Emergent Patterns of Options." As it became clear in the course of using SIBYL, the use of a semi-formal representation requires the user to understand many assumptions that are not captured explicitly in the representation itself. A contribution of the QOC research is to make clear the importance of making explicit such assumptions underlying the use of a rationale representation language.

QOC is yet a representation to be used with a paper and a pencil; as such, no *computational operations* have been defined on QOC. Thus, the rationales generated by QOC are potentially reusable, but there is no discussion of how relevant rationales are to be determined and retrieved. In the absence of a relevance metric and the computational support for its application, reuse be difficult to achieve when the rationales accumulate beyond a small scale. Given QOC's explicit representation of the criteria, a goal lattice like the one used for SIBYL can be used for determining relevance. However, it would be interesting to see how QOC addresses this problem in the future or in general what computational operations are defined on it, and compare with those for SIBYL.

Given the ambiguity about what exactly constitutes the vocabulary of QOC, however, QOC seems currently more a model rather than a fully-developed representation language. That is, it seems to be an attempt to understand and categorize the elements of design rationale without providing yet a specific vocabulary for expressing them. This observation is also consistent with the authors' warning against premature commitment to a specific representation [MacLean et al. 1989]. Considering the similarity between QOC and DRL in the underlying structure, we hope that DRL provides a representation language adequate for representing most of the elements that the QOC research has been articulating.

Chapter 8

Conclusions

This chapter summarizes the contributions of this research and discusses the topics for future research.

8.1 Contributions

This thesis has explored the feasibility and the benefits of a system that captures and manages rationales for decisions, in which the domain knowledge is too expensive or not possible to formalize. This exploration produced the following contributions:

- *identification and categorization of the needs for managing rationales.* The needs for explicit representation and management of decision rationales were identified and classified into four categories (Section 1.1): managing rationales within a session, managing rationales across sessions, reusing rationales from past decisions, sharing rationales across decisions.

- *articulation of the concept of a decision rationale management system.* A scenario was constructed that illustrates ways of managing rationales (Chapter 2). This scenario was used to identify the components needed for successful rationale management: the language for representing the elements of rationales, the method for using the language to capture the rationales, and the services that use the captured rationales to support decision making. In order to characterize systems with these components, the concept of a decision rationale management system was proposed. Existing decision rationale management systems and their scopes were characterized by categorizing them along two dimensions: formality of the representation used and reusability (Chapter 7).

- *characterization of the structure of decision rationales.* In order to design a decision rationale management system that supports the tasks illustrated in the scenario, I developed a sequence of models that progressively differentiates the elements of decision rationales needed to support the tasks (Chapter 3). Decision rationales were characterized in terms of five spaces: the argument, the alternative, the evaluation, the criteria, and the issue spaces. Starting with the argument space, each model next in the sequence makes an additional space explicit. I have discussed what extra tasks this additional differentiation can support.

- *the design of DRL, a language for representing decision rationales.* In order to provide a language for rationale management, I have developed DRL (Chapter 4), which provides constructs for representing the elements in the five spaces that characterize the structure of decision rationales.

- *the design and implementation of SIBYL, a decision rationale management system that demonstrates the feasibility and the benefits of capturing and managing decision rationales without requiring the formalization of domain knowledge.*

In particular,

- *the implementation of DRL in such a way that rationales can be captured and managed without the formalization of the domain knowledge.* SIBYL achieves this task by implementing the constructs of DRL semi-formally, i. e. as formal objects whose attributes values can be informal descriptions as well as formal objects (Chapter 5).

- *the design and implementation of an interface that embodies a method for using DRL.* In order to provide an environment in which DRL can be used to capture rationales with as little cost as possible, I have designed and implemented the features -- such as decision matrices, submatrices, argument browsers, and context-sensitive menus -- that help people easily see relevant parts of the rationales, examine details at various levels, structure their arguments, and find out appropriate actions to take.

- the design and implementation of *a set of computational services that use the captured rationales to provide decision support as well as the motivation for people to use SIBYL to represent rationales* (Chapter 6). The precedent management demonstrates that rationales can be reused, through human interaction, even in non-formalized domains (6.1). I have also implemented the dependency management (6.2) and the viewpoint management (6.3) that meet at least the minimal needs that are crucial in managing decision rationales: maintaining dependencies on the one hand and keeping track of multiple decision states on the other. Also, SIBYL uses the Object Lens features such as agents, the rule system, and the different display formats to help the user to monitor for a decision state, better grasp the relations among the elements of the captured rationales.

8.2 Future Research

There are three directions in which I want to push ahead this research . First, I would like to test and extend the adequacy of DRL as well as the SIBYL environment as a whole by subjecting them to larger scale uses. Secondly, I would like to explore the different ways of giving more knowledge to SIBYL, in particular the aspect of incremental formalization, as discussed in Chapter 8. Thirdly, as a long term goal, I would like to explore ways to automate the parts of SIBYL and articulate a computational problem solving paradigm based on arguments through incremental formalization.

Test and Extension

I would like to test and extend the adequacy of the current representation and the system by having it used on larger scale examples. So far, DRL and SIBYL have been used by a small group (4 to 6 people) on small scale problems such as laying out floor space and designing user interfaces. Several cases, such as designing a window manager or choosing a hardware platform, have been reconstructed from published cases. DRL has evolved over time to incorporate lessons from these experiences. It is clear, however, that a large scale use on more complex problems will bring in additional or different sets of constraints. These constraints will undoubtedly lead to modification and extension of the current representations and services. I would like to test how well DRL and SIBYL holds up in larger scale uses.

Before SIBYL can be taken to sites, however, there are several steps that need to be taken beforehand.

- The system needs to be free of bugs and made faster. People are well known not to put up with any inconvenience when the benefits they get are not clear or immediate. If there is any hope of getting it used at all, then it must be bug-free and efficient.

- The system needs to be integrated with the other aspects of the task it will support. For example, if the task is decision making by stockbrokers, for example, SIBYL should be able to interact with the Dow-Jones database. Or if SIBYL is used for software design, then people should be able to use the requirements collected in the requirement analysis phase as goals or at least refer to them. Likewise, designers should also be able to refer to other forms, such as drawings, that they produce

throughout out the design process. This integrated environment is important both because people do not want to duplicate information that they already provided and also because switching contexts interrupts the work flow.

- The system needs to provide a better support for evolution. In real-life decision making, goals change, arguments need to be reformulated, and alternatives need to be grouped in different ways. SIBYL maintains, at least in principle, that no DRL object can be modified (as opposed to augmented) in order to keep the record of the evolution as well as to make sure that there is no dangling object, i.e. those objects whose validity assumes the existence of another object, which is now gone or different from what it was. Instead, SIBYL deal with these changes is through Viewpoints. When goals change, for example, we capture the existing state as a Viewpoint, create a new Viewpoint as a copy of the existing state, and then in the new viewpoint change the goals. The use of these viewpoints allow us to accommodate changes while keeping a record of the evolution. However, the grain size of the viewpoint is too large. If we insisted on the non-modifiability principle, viewpoints would proliferate because changes including ones due to mistakes happen very often. Hence, there needs to be a balance between non-modifiability and non-proliferation, and it is not clear yet what mechanism would achieve that balance.

Giving More Knowledge to SIBYL

The only kind of knowledge that SIBYL has so far is that generic to the task of decision making. This task-specific knowledge, furthermore, is either built into the types and their relations of DRL or built into the routines that manipulated them. SIBYL has no

understanding of the domain knowledge, that is no knowledge about the objects and their relations in the domain except as DRL constructs. For example, SIBYL did not understand what a window manager is except as an instance of Alternatives. Therefore, the reasoning power of SIBYL was limited.

The decision to give SIBYL only the task-specific knowledge was deliberate. A goal of this research was to explore the possibility and the power of a semi-formal system [Lai et al. 1988]. That is, I wanted to explore a system which is useful even when its input is completely informal (i.e. known to the system only as text-strings) albeit combined with varying degrees of formalized knowledge to be exploited in order to provide more powerful services. There are at least three reasons why it is important to explore semi-formal systems. It allows us to capture and manage rationales in tasks for which the domain knowledge is too expensive for the benefits sought, in tasks for which the domain knowledge is currently not available, and in tasks for which common-sense reasoning is needed. Incremental formalization made possible by a semi-formal system would allow us to formalize the knowledge as the needs arise, as only the benefits of doing so overrides the cost, and as we gain more understanding of the domain.

In this thesis, I regard my contribution as having laid out a foundation for exploring these issues, and cannot really claim to have explored all the possibilities and the powers of incremental formalization. Nevertheless, the experience with SIBYL so far does illustrate many of these benefits. In particular, I describe three ways in which to make more knowledge accessible to SIBYL: formalizing informal into existing formal constructs, extending the DRL vocabulary, and formalizing the domain knowledge. An example of the first is given in Chapter 5, where the evaluation of a claim was originally a text string (HIGH, Susan), but it changed to an instance of a claim, [HIGH] , when there was a need to formalize, for example to argue about it. An example of the second is given by the

extension of DRL to capture *design rationales*. This extension is discussed in [Lee 1991]. With a few additional constructs, such as Artifact and Derives, that captures the knowledge about design decisions and the resulting artifacts, SIBYL can, for example, provide services that help the user to look for artifacts which use a specified set of alternatives and see how their design decisions turned out.

An example of the third, i.e. formalizing the domain knowledge, is given by mSIBYL [Ruecker & Seering 1990], whose goal is to extend SIBYL to the task-domain of mechanical engineering design. In mSIBYL, informal descriptions of the goals and the alternatives are turned into formal descriptions using the transition space representation [Borchardt 1990]. This formal description, still coexisting with other informal descriptions (such as of claims), provides the system with more knowledge about the behaviors of the artifact desired and the alternatives available so that it can retrieve more relevant precedents with less interaction with the user. These examples, however, illustrate only the beginning of an exploration of incremental formalization. I, in collaboration with others, would like to continue this exploration. In particular, I would like to study how much formalization is required in what way to transfer the research on precedent-based learning [Winston 1982] and case-based reasoning to rationale management research.

Toward a Dialectical Problem Solving Paradigm

One of the motivations underlying the current research, though still far away to be realized, was to develop a computational problem solving paradigm based on arguments. The importance of 'deliberate' decision process in automated reasoning -- in the sense of involving arguments pro and con, evaluating alternatives, compromising and negotiating --

has been appreciated only recently by the realization that a strictly deductive approach does not capture "the openness" of realistic systems. What really happens in real systems and what needs to be included in artificial systems is that problems are solved through the dialectical processes involving the interaction among agents with different goals, shared resources, and different expertise.

Identifying and articulating the objects and the processes involved in deliberate decision making, as this work tries to do, is a small but an important step in the study of such dialectical processes. There is also a good body of research on these topics [Bond & Gasser 1988; Doyle 1980; Kornfeld & Hewitt 1981; Minsky 1987; Rescher 1977] I hope that while exploring incremental formalization, I can gain more understanding of how this research is related to this body of research, and make a contribution toward a dialectical problem solving paradigm.

Appendix: Details of DRL

This appendix is a reference manual for DRL, which provides a description of each of its constructs, including its semantics. First, I give a brief overview, followed by the description of the constructs in alphabetical order.

The semantic classes for DRL are:

STATE

ACTION

STATEMENT

QUESTION

PROCEDURE

A *Goal* represents a desired STATE. An *Alternative* is a possible ACTION that can be taken. A *Claim* is a STATEMENT. A relation in DRL is a CLAIM. For example, *Supports (C1, C2)*, where C1 and C2 are claims, is a STATEMENT about the relational state between two STATEMENT's. *Achieves (A, G)* is a STATEMENT about the relational state between the state produced by applying the alternative action, A, on the current state and the goal state, G.

It should be noted that users often give names to DRL objects, such as "Which application interface for the window manager, WING," or "Interaction Manager," which suggest wrong semantics. These labels are of course mere text strings for the computer, and their primary purpose is for human understanding and communication of the rationales. For computational purposes, though, their semantics do not depend on the labels. To illustrate this with our example decision, the decision problem, "Which application interface for our window manager, WING" is to be interpreted as the desired state of having chosen a good alternative. Its subgoals, "Highly portable" or "Common interface," are to be interpreted as the state of having chosen an alternative that helps the window manager to be portable or that helps the window manager to promote a common interface. And a *Is A Subgoal Of (G1, G2)* should be interpreted as a statement that the state represented by G1 is a part of achieving the state represented by G2. For example, "Support Direct Manipulation" is a subgoal of "Ease of Use for Naive Users," means that the state of having chosen an alternative that will help support direct manipulation is a part of achieving the state of having chosen an alternative that will build applications that are easy to use for naive users.

Alternatives, like "Interaction Manager," are to be interpreted as actions, like using interaction manager as the application interface. An achieves relation, like "Interaction Manager achieves the goal of being portable," is interpreted as a statement about the relation between an alternative action taken, e.g. building a window manager using an interaction manager as the application interface, and the state represented by the goal of being portable. Thus, the interpretation of the achieves relation is the statement that the action of using an interaction manager as the application interface for WING achieves the state of having chosen an alternative that helps the window manager to be portable.

In the following, I describe for each construct of DRL its semantics, its intended use, and the important attributes. For relational constructs, the argument types are shown in the parenthesis. The attributes, Creator and Creation Date, which represent respectively the user who creates its instance and the date of creation, are present in all of the constructs and not discussed further.

Achieves (Alternative, Goal)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes: Degree, Plausibility, Evaluation

Achieves is a statement about how close the state produced by applying an alternative action to the current state is to a goal state. Any statement is expressed as a *Claim* in DRL, which provides no built-in distinction between facts, beliefs, and opinions (cf. Claim). Achieves (A, G) is interpreted as the claim that the alternative A achieves the goal G. As a claim, of course, it can be argued about. In particular, the arguments for the evaluation of an alternative are represented as claims supporting, denying, or qualifying (i.e. presupposed by) an achieves relation between the alternative and an appropriate goal. The Degree attribute represents the extent to which the alternative achieves the goal. The Plausibility attribute represents the measure of how plausible that measure is. The Evaluation attribute of an achieves relation represents the measure of how the alternative achieves the goal, which is typically the degree modulated by the plausibility measure. (cf. *Claim)*

Alternative

Semantic Class: ACTION

Supertype: *DRL Object*

Attributes: Status

Alternative represents option in consideration, e.g. "Interaction Manager" or "Toolkit". An option is an action, e.g. "Using Interaction Manager," although their labels often mention only the objects involved (like "Interaction Manager"). Alternatives that specialize a given alternative are related to the latter via *Is A Kind Of* relations. For example, 'Extensible Interaction Manager' is *Is A Kind Of* 'Interaction Manager'. The Status of an Alternative indicates the current information about the alternative, e.g. Chosen or Inactive. An alternative is related to each goal via an *Achieves* relation, representing the claim that the alternative achieves the goal (cf. *Achieves*). An alternative is related to the decision problem for which it is an option via an *Is A Good Alternative For* relation. The Evaluation attribute of this relation represents the overall measure of how well the alternative satisfies the subgoals (or the requirements) of the decision problem (cf. *Is A Good Alternative For*).

Answers (Claim, Question)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes: Degree, Plausibility

Answers is a statement about the relation between claim C and question Q. An answer to a question is represented in DRL as a *Claim* which is related to the question via this relation.

There may be several competing answers to a question, each of the claims representing an answer would then be related to the question via the answers relation. One can argue about which answer is more likely by arguing about the corresponding answers relation. The Degree attribute represents the extent to which the claim answers the question. The Plausibility attribute represents the measure of how plausible the answer is.

Claim

Semantic Class: STATEMENT

Supertype: *DRL Object*

Attributes: Plausibility, Degree, Evaluation

Claim is used to represent any statement, including facts, arguments, assumptions, answers, and comments. DRL does not provide separate constructs for these categories of statements; instead, it allows the distinction between them to be captured dynamically. A fact is a claim whose plausibility is very high, beyond a threshold that can be set by the user. An assumption is a claim which is assumed to be true hypothetically for the time being, i.e. within a prescribed context (cf. *Viewpoint).* Any answer or a comment can be represented as a claim. What makes a statement an answer or a comment is nothing in the statement itself but its role, i.e. the relation that it has to other objects (cf. *Answers, Comments).* The truth of any claim, whether it represents a fact, an assumption, or an answer, can be argued about, i. e. can be related to another claim through *Supports*, *Denies*, or *Presupposes* relations.

The Plausibility attribute of a claim represents the measure of how likely the claim is true. The term, *plausibility*, is used instead of *probability* because we do not want to restrict the

measure used for the attribute to the mathematical probability. It could be nominal categories, like Highly Plausible, Moderately Plausible, and Implausible.

The Degree attribute of a claim represents the degree to which the claim is true. For example, the degree of the claim, "Interaction Manager is expensive to build," may be represented by a range of money that it will typically cost to build an interaction manager. The plausibility of a claim often depends on the degree of the claim. That is, we can only assess how likely the claim is true when the degree is made explicit. For example, it is difficult to assess "Interaction Manager is expensive to build" without knowing what the claim means by being expensive. Strictly speaking, therefore, the degree of a claim should be an explicit part of the claim, e.g. "Interaction Manager costs around $ 40,000 plus minus $ 5,000."

Nevertheless, the Degree of a claim was made a separate attribute for the following reasons. First, the degree of a claim changes as other claims produced in the course of decision making. For example, the degree of an Achieves relation reflects the extent to which an alternative achieves a goal, and it changes as claims are produced that support or deny it. In fact, the degree of any claim is subject to change as claims are produced that support or deny it. Having Degree as a separate attribute allows this measure to be changed without having to create a separate statement every time the measure changes. Second, it is often difficult to specify the degree of a claim at the time that the claim is made. For example, some claims, like an Achieves(A, G), are created automatically, and it is not clear what its degree should be. It is not 100% because that would rule out the cases where an alternative would achieve only some part of the goal and most claims would deny such a claim. It is not 0% because any alternative worth considering would likely achieve some part of the goal. Even for those claims that the human user produces, specifying their exact degrees may require too much efforts and much arbitrariness. It seems that a

qualitative statement such as "IM is expensive to build" has some implicitly agreed-upon degree so that people can argue about it without requiring its degree to be explicit.

Of course, we should make sure that when the degree of a claim changes, other claims that were produced about it are still relevant because they were responding to a different claim, , strictly speaking,. Hence, when the degree or the degree interval that was assumed when claims were made changes, the decision makers should be notified so that the existing arguments can be modified accordingly. Of course, there may be practical problems in doing that. Those who produced the argument may not remember the situation or may be gone. This would be a serious problem if the evaluation was to be computed automatically, and this is one of the reasons why it is so difficult to produce automatic evaluations. For human users, the problem is less serious because the arguments that people put forward seem mostly qualitative and do not usually depend heavily on the quantitative degree of the statement they are about. A possible exception is if the degree of the statement goes radically out of reasonable bounds, e.g. fast hardware means 2000 mips.

The Evaluation attribute is used to represent a summary evaluation measure that typically combines the effect of the degree and the plausibility of a claim. For example, take the claim that using a good debugging environment reduces the development cost. Let's say that its plausibility is 80%, its degree is 40, 000 dollars. Its Evaluation is whatever they mean overall to the user, e.g. 40,000 * .8. This purpose of this attribute is to provide a single dimension along which the claims can be compared.

Comments (Claim, DRL Object)

Semantic Class: Statement

Supertype: *Suggests*

Attributes: Creator, Creation Date,

Comments is a statement about a DRL object. It is used to represent a statement about an object, which one does not want to be interpreted as anything more specific, like *Supports or Denies* . *Comments (C, O)* is always true by the mere fact that it is created, although the truth of the claim C is subject to arguments.

Decided

Semantic Class: STATE

Supertype: *Status*

Attributes: Chosen Alternative, Rationales

Decided is the state of having chosen an alternative for a decision problem. It is used to represent the fact that the decision has been made together with other information about the decision made. It is a subtype of *Status*. The Chosen Alternative attribute represents the alternative chosen. The Rationales attribute represents the reasons for having chosen the alternative, which may be expressed either in free-text or with pointers to the DRL objects responsible for the final decision.

Decision Problem

Semantic Class: STATE

Supertype: *Goal*

Attributes: Status, Evaluation

Decision Problem represents the desired state of having chosen the best alternative. A decision problem is a special kind of *Goal* representing the goal of choosing the optimal alternative. At the intuitive level, a decision problem represents the problem of choosing the alternative that best satisfies its subgoals. These subgoals elaborate in turn what it means to satisfy the top level goal. For example, "Reduce Development Cost" and "Is Portable" are two subgoals of the Decision Problem, "Which application interface for our window manager, WING?" These subgoals are to be interpreted as the desired states of having chosen an alternative that satisfies the requirement that they stand for.

The Status attribute of a decision problem represents the current status of the decision, e.g. Resolved, Unresolved, Waiting for Further Information. The status can be an object of its own right, for example, *Decided* is a subtype of *Status* with a field called Rationale, which point to the objects that were directly responsible for the final decision outcome. The Evaluation field records the evaluation of the decision process, e.g. success or failure.

Denies (Claim, Claim)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes: Degree, Plausibility, Evaluation

Denies is a statement about the relation between the evaluation attributes of the two statements. It is used to represent an argument that refutes another argument. *Denies (C1, C2* is true if the evaluation of C2 is a monotonically decreasing function of the evaluation of C1. (cf. *Claim)*

Goal

Semantic Class: STATE

Supertype: *DRL Object*

Attributes: Importance

A *Goal* represents a state that one wants to achieve by making a decision. A goal serves as a criterion for evaluating alternatives. A goal may have subgoals, which are the states, once achieved, that help bring about the state represented by the parent goal. These subgoals are related to the parent goal via the *Is A Subgoal Of* relation (cf. *Is A Subgoal Of).* A goal may specialize another goal, e.g. "Minimize Development Cost," specializes "Minimize Cost." The specialization relationship is represented with the *Is A Kind Of* relation. Other relations among goals, such as whether the subgoals are conjunctive (i.e. all of them have to be satisfied to achieve the parent goal) or disjunctive (i.e. (i.e. satisfying one of them makes it unnecessary to satisfy the others in the set), can be represented with the construct, GROUP. (cf. *Group)* The Importance attribute of a goal is used to represent how important the goal is.

Group

Semantic Class: A Set of Semantic Class of its Members

Supertype: *DRL Object*

Attributes: Members, Member Relationship

A *Group* is used to group a set of objects of the same type among which we want to indicate some relationship. The attribute, Members, points to the objects to be grouped, and the relation among them -- e.g. Conjunctive, Disjunctive, Mutually Exhaustive -- are represented in the Member Relationship attribute.

Is A Good Alternative For (Alternative, Decision Problem)

Semantic Class: STATEMENT

Supertype: *Achieves*

Attribute: Degree, Plausibility, Evaluation

Is A Good Alternative For is a statement about the relation between the action represented by the alternative and the goal state represented by the decision problem. It is an assertion that taking the action achieves the goal state DP. It is a special case of *Achieves (A, G)* where G is restricted to a goal of the type *Decision Problem.* Its Degree attribute therefore represents the measure of how good the alternative A is with respect to the overall goal of choosing the best alternative, and its value is a function of the evaluations of the achieves relations between the alternative and all the subgoals for the decision problem. The Plausibility attribute represents the measure of how convincing the degree assignment is.

The Evaluation attribute represents the overall evaluation of the alternative which takes into account both the degree and the plausibility. (cf. *Claim*)

Is A Kind Of (DRL Object, DRL Object)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attribute: Degree, Plausibility. Evaluation

Is A Kind Of is a statement about the relation between two DRL objects. It is used to represent the specialization/generalization relationship among two objects. *Is A Kind Of (O1, O2)* is true if the extension of O1 subsumes the extension of O2 (Schmolze ..), although the automatic classification based on subsumption is not usually possible in SIBYL because of the informally specified attributes.

Is An Answering Procedure For (Procedure, Question)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attribute: Degree, Plausibility

Is An Answering Procedure For is a statement about the relation between a procedure and a question. It is used to represent a procedure used to answer a question. *Is An Answering Procedure For (P, Q)* is true if P is the right procedure for answering the question Q. The Degree attribute represents the extent to which the procedure is appropriate for answering

the question. The Plausibility represents the uncertainty about the appropriateness specified in the degree attribute.

Is A Result Of (Claim, Procedure)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attribute: Degree, Plausibility

Is A Result Of is a statement about the relation between a claim and a procedure. It is used to represent the procedure used for obtaining the answer which in turn is represented as a claim. It is true if C is the assertion that represents the right interpretation of the result of correctly running the procedure P. The Degree attribute represents the extent to which the claim C is in fact based on the procedure P. The Plausibility attribute represents the uncertainty about the extent specified in the degree attribute.

Is A Subdecision Of (Decision Problem, Decision Problem)

Semantic Class: STATEMENT

Supertype: *Is A Subgoal Of*

Attribute: Degree, Plausibility

Is A Subdecision Of is a statement about the relation between two decision problems. It is used to represent a decision problem that needs to be resolved in order to resolve another decision problem. *Is A Subdecision Of (DP1, DP2)* is true if achieving the goal state

represented by DP1 requires achieving the goal state represented by DP2. It is a specialization of *Is A Subgoal Of (G1, G2)* where G1 and G2 are restricted to be of type, decision problem (cf. *Is A Subgoal Of*). The Degree attribute represents the extent to which resolving DP2 is necessary for resolving DP2. The Plausibility attribute represents the uncertainty about the extent specified in the degree attribute.

Is A Subgoal Of (Goal, Goal)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attribute: Degree, Plausibility

Is A Subgoal Of is a statement about the relation between two goals. It is used to elaborate a goal in terms of more specific goals that together will achieve the original goal. It is also used to claim that a goal needs to be achieved in order to achieve another goal. *Is A Subgoal Of (G1, G2)* is true if achieving the goal state represented by G2 requires achieving the goal state represented by G2. The Degree attribute represents the extent to which G1 needs to achieved in order to achieve G2. The Plausibility attribute represents the uncertainty about the extent specified in the degree attribute.

Is Related To (DRL Object, DRL Object)

Semantic Class: STATEMENT

Supertype: *Claim*

Attributes: Degree, Plausibility

Is Related To is a statement about the relation between two DRL objects. It is used to represent a relation that are not captured by any of its subclasses. It also serves as a parent of all DRL relations, from which these relations inherit attributes that are specific to all the relations. It is also a subclass of *Claim*. Hence, all DRL relations are subclasses of CLAIM. For example, "*G1 Is A Subgoal Of G2*'" is interpreted as the claim that G1 should be a subgoal of G2'. As such, any relation can be supported, refuted, qualified, or questioned just like any other claims. The Degree attribute represents the extent to which the two objects are related, and the Plausibility attribute the uncertainty about the extent specified in the Degree attribute.

Procedure

Semantic Class: PROCEDURE

Supertype: *DRL Object*

Attributes: Steps

Procedure is used to represent a procedure, for example which has been used to answer a question. Procedure represented may be executable directly or need to be interpreted by human users. Information about the procedure used can be useful later when the answer needs to be rechecked or when a similar question has to be answered. The Steps attribute represents individual steps that make up a procedure. Each of these steps can be a procedure or an informal description.

Presupposes (Claim, Claim)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes: Degree, Plausibility

Presupposes is a statement about the relation between two claims. It is used to specify a condition under which a claim would be true. *Presupposes (C1, C2)* is true if C1 is true only if C2 is true. Sometimes, *Qualifies (C1, C2)* is used as an alisas for *Presupposes (C2, C1)*. The Degree attribut represents the extent to which the truth of C1 depends on the truth of C2. The Plausibility attribute represents the uncertainty about the degree specified in the degree attribute.

Question

Semantic Class: QUESTION

Supertype: *DRL Object*

Attributes: Deadline

A *Question* is used to request for more information. An answer to a question is represented as a Claim, which is linked to the question via an Answers relation. A question may be also related to a procedure via *an Is An Answering Procedure For* to indicate a procedure that can be used to answer the question. The answer produced by executing the procedure is then linked to the procedure via the relation, *Is A Result Of*. The answer is related to the

question via the relation, *Answers*. The Deadline attribute indicates the date before which the answer is wanted.

Raises (DRL Object, Question)

Semantic Class: Statement

Supertype: *Suggests*

Raises is a statement about the relation between a DRL object and a question. It is used to indicate that the question arose as a result of examining an DRL object. It is a special case of *Suggests (O1, O2t)*, where O2 is restricted to the type, Question.

Status

Semantic Class: STATE

Supertype: *DRL Object*

Attributes:

Status is used to indicate a state in which an object is in. Many objects in DRL have the Status field, which provides information about the state that the object is in. For example, Decision Problems may have been *Decided, Active, Inactivated, or Archived.* Alternatives may be been *Chosen, Rejected, or Inactivated.* Questions may have been *Answered or Being Answered.* Usually, simple keywords indicating these status are sufficient. However, in some cases, these status have internal structures. For example, if a decision problem has been decided, it is important that a record is made of the person who made the

decision, the date of the decision, and the major reasons for the decision. Or if a question is *Being Answered,* it should provide information about who is in the process of answering the question and how she is planning to do so. In these cases, one creates an instance of *Status*, and fill it with appropriate information. An example of a built-in status is *Decided.*

Supports (Claim, Claim)

Semantic Class: STATEMENT

Supertype: *Claim*

Attributes: Degree, Plausibility

Supports (C1, C2) is a statement about the relation between the evaluation attributes of the two statements, C1 and C2. It is used to represent an argument that supports another argument. *Supports (C1, C2* is true if the evaluation of C2 is a monotonic function of the evaluation of C1. (cf. *Claim)*

Suggests (DRL Object, DRL Object)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes:

Suggests is a statement about the relation between two DRL objects. It is used to represent the fact that an object was created as a result of examing another object. *Suggests (O1, O2)* is true if O2 is in fact created as a result of examining O1.

Tradeoffs (DRL Object, DRL Object, attribute)

Semantic Class: STATEMENT

Supertype: *Is Related To*

Attributes: Degree, Plausibility

Tradeoff is a statement about the relation between two DRL objects concerning a given attribute. It is used to assert that increasing a given attribute value for an object decreases the value of the same attribute for another object. *Tradeoff (O1, O2, A)* is true if the attribute A is of an ordinal scale and the value of the attribute A of O1 is a monotonically decreasing function of the value of the attribute A of O2.

Viewpoint

Semantic Class: STATE

Supertype: *DRL Object*

Attributes: Elements, View Relations

Viewpoint is used to represent a state of decision making process that needs o be captured for comparison or archival purposes. For example, the decision states that a viewpoint can represent include those:

- that have the same set of objects but with different attributes.
- that have objects that are subset or superset of the objects in the other viewpoints.
- that are historically related, i.e. ones that have been generated from another in the class

The Elements attribute points to all the objects as well as their relations that belong to that decision state. The View Relations tells us all the relations between this viewpoint and other viewpoints (Cf. Section 6.3).

References

Bobrow, D. & I. Goldstein (1980). Representing Design Alternatives. *Proceedings of Artificial Intelligence and Simulation of Behavior* Amsterdam, Netherlands.

Boehm, B. (1981). *Software Engineering Economics.* Prentice Hall: Englewood Cliff, N.J.

Bond, A. H. & L. Gasser (Eds.) (1988). *Readings in Distributed Artificial Intelligence.* vol. Morgan Kaufmann San Mateo, CA.

Borchardt, G. (1990). Transition space. AI Memo 1238, MIT: Cambridge, MA

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. in Michalski, R. S., J. G. Carbonell & T. M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach.* vol. 2 pp. 371-392. Morgan Kaufmann: Los Altos, CA

Conklin, J. & K. C. Yakemovic (in print). A Process-oriented Paradigm for Design Rationale. to appear in *Human-Computer Interaction* the special issue on design rationale

Conklin, J. & M. L. Begeman (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems* 6(4) pp. 303-331

Davis, R. & D. Lenat (1982). *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill: New York

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence Journal* 28(2) pp. 127-162

DeSanctis, G. & R. B. Gallupe (1987). A foundation for the study of group decision support systems. *Management Science* 33(5) pp. 589-609

DeSanctis, G., V. Sambamurthy & R. T. Watson (1987). Computer-suppoted meetings: building a research environment. *Large Scale Systems* 13(1) pp. 43-59

Doyle, J. (1980). *A Model for Deliberation, Action, and Introspection.* MIT Artificial Intelligence Laboratory,

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence Journal* 12(3) pp. 231-272

Duda, R. O., P. E. Hart & N. Nilsson (1976). Subjective Baysian methods for rule-based inference system. in [Webber & Nilsson 1981], pp. 192-199

Faulkenhainer, B. (1989). The structure mapping engine: algorithm and examples. *Artificial Intelligence Journal* 41(1) pp. 1-64

Fischer, G., R. McCall & A. Morch (1989). Design Environments for Constructive and Argumentative Design. *Proceedings of CHI'89* pp. 269-276. Austin, TX.

Greene, K. J., M. Bouler, S. Gerhart, D. M. Russinoff (1991) Spectra 0.1: demonstration and research issues. MCC Technical Report STP/EI-329-90 (videotape), MCC Software Technology Group: Austin, TX

Gray, P. (1987). Group decision support systems. *Decision Support Systems* 3(3) pp. 233-242

Hall, R. (1989). Computational approaches to analogical reasoning: a comparative analysis. *Artificial Intelligence Journal* 39 pp. 39-120

Hopgood, F. R. A., D. A. Duce, E. V. C. Fielding, K. Robinson & A. S. Williams, (Eds.) (1986). *Methodology of Window Management.* Springer-Verlag: New York.

Huber, G. (1984). Issues in the design of group decision support systems. 8(3) pp. 195-204

Huhns, M. & R. D. Acosta (1988). ARGO: A System for Design by Analogy. *IEEE Computer* (Fall, 1988) pp. 53-68

Kahneman, D., P. Slovic & A. Tversky (Eds.) (1982). *Judgement under Uncertainty: Heuristics and Biases.* Univ. of Cambridge Press Cambridge, England.

Kleimuntz, B. (1990). Why we still use our heads instead of formulas: toward an integrative approach. *Psychological Bulletin* 107(3) pp. 296-310

Kolodner, J., Eds. (1988). *Proceedings of DARPA Workshop on Case-Based Reasoning.* Clearwater Beach, FL

Kornfeld, W. A. & C. Hewitt (1981). The scientific community metaphor. *IEEE Trans. Syst. Man Cyber.* v. 11 pp. 24-33

Kraemer, K. & J. L. King (1988). Computer-Based Systems for Cooperative Work and Group Decision Making. 20(2) pp. 115-146

Kunz, W. & H. Rittel (1970). Issues as Elements of Information Systems. Center for Planning and Development Research Univ. of California, Berkeley, Working Paper 131

Lai, K. (1991). Object Lens: User Guide. MIT Center for Coordination Science: Cambridge, MA

Lai, K.-Y., T. Malone & K.-C. Yu (1988). Object Lens: A "Spreadsheet" for Cooperative Work. *ACM Transactions on Office Information Systems* 6(4) pp. 332-353

Lane, T. (1990). User Interface Software Structures. Tech Report CMU-CS-90-101 Ph. D. thesis in Computer Science Department. CMU: Pittsburgh, PA

Lee, J. (in press). What's in Design Rationale? to appear in *Human-Computer Interaction* (Fall, 1991), the special issue on design rationale

Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. *Proceedings of 13th International Conference on Software Engineering* pp. 114-125 Austin, TX.

Lee, J. (1990). SIBYL: A Qualitative Decision Management System. in Winston, P. H. & S. Shellard (Eds.) *Artificial Intelligence at MIT: Expanding Frontiers.* vol. 1. Chapter 5 MIT Press: Cambridge, MA

Lee, J. (1990). SIBYL: A Tool for Managing Group Decision Rationale. *Proceedings of Computer Supported Cooperative Work* pp. 79-92 ACM Press: LA, CA

Lee, J. & T. Malone (1990). Partially Shared Views: A Scheme for Communicating among Groups that Use Different Type Hierarchies. *Transactions on Information Systems* 8(1) pp. 1-26

Lee, J. (1989). Task Embedded Knowledge Acquisition through a Task-Specific Language. *Proceedings of IJCAI Workshop on Knowledge Acquisition* Detroit, MI.

Lee, J. & T. Malone (1988). How Can Groups Communicate When They Use Different Languages? Translating Between Partially Shared Type Hierarchies. *Proceedings of Proceedings of the ACM Conference on Office Information Systems* pp. 22-29. ACM Press: Palo Alto, CA

Lee, J. & K.-Y. Lai (1991). A comparative analysis of design rationale representations. CCS Tech Report #121 MIT Center for Coordination Science: Cambridge, MA

Lubar, M. (1991). Representing design dependencies in an issue-based style. *IEEE Software* (July) pp. 81-89

MacLean, A., R. M. Young & T. P. Moran (1989). Design Rationale: The Argument behind the Artifact. *Proceedings of CHI'89* Austin, TX.

MacLean, A., R. Young, V. Bellotti & T. Moran (in press). Questions, Options, and Criteria: Elements of a Design Rationale for User Interfaces. to appear in *Human-Computer Interaction* the special issue on design rationale

Mark, W. & J. Schlossberg (1990). Design memory. *Proceedings of Knowledge Acquisition for Knowledge-Based Systems Workshop* Banff, Canada.

McCall, R. (1987). PHIBIS: Procedurally Hierarchical Issue-Based Information Systems. *Proceedings of Conference on Planning and Design in Architecture* Boston, MA. American Society of Mechanical Engineers

McKinlay, J., S. Card & G. Robertson (1990). A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction* 5(2-3)

Minsky, M. L. (1987). *The Society of Mind.* Simon and Schuster: New York

Mostow, J. (1989). Design by Derivational Analogy: Issues in the Automated Replay of Design Plans. *Artificial Intelligence Journal* 40 pp. 119-184

Mostow, J. (1986). Why are design derivations hard to replay? in Mitchell, T., J. Carbonell & R. Michalski, Eds. (1986). Machine Learning: A Guide to Current Research. vol. Kluwer Hingham, MA. (Eds.) .

Mostow, J. (1985). Toward Better Models of the Design Process. *AI Magazine* 6(1) pp. 44-57

Mostow, J. & M. Barley (1987). Automated reuse of design plans. *Proceedings of 1987 International Conference on Engineering Design* pp. 632-647. Boston, MA.

Nunamaker, J., A. R. Dennis, J. S. Valacich, D. R. Vogel & J. F. George (1991). Electronic meeting systems to support group work. *Communications of ACM* pp. 40-61

Nunamaker, J. F. J., L. M. Applegate & B. R. Konsynski (1988). Computer-aided deliberation: model management and group decision support. *Operations Research* 23(6) pp. 826-848

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann: San Mateo, CA

Pitz, G. F. & N. J. Sachs (1984). Judgement and decision: theory and application. in *Annual Review of Psyc ..logy.* vol. 35 pp. 139-163.

Potts, C. & G. Bruns (1988). Recording the Reasons for Design Decisions. *Proceedings of 10th International Conference on Software Engineering* pp. 418 -427.

Rescher, N. (1977). *Dialectics: A Controversy-Oriented Approach to the Theory of Knowledge.* State Univ. of New York Press:

Riesbeck, C. K. & R. C. Schank (1989). *Inside case-based reasoning.* Lawrence Erlbaum Associates: Hillsdale, N. J.

Reucker, L. & W. P. Seering (1991). A dialectical reasoning system for design documentation. *Proceedings of the 3rd ASME Conference on Design Theory and Methodology* Miami, FL.

Schmolze, J. G. & T. A. Lipkis (1983). Classification in the KL-ONE knowledge representation system. *Proceedings of IJCAI-83* Karlsruhe, West Germany.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press: Princeton, N.J.

Shafer, G. & R. Logan (1989). Implementing Dempster's rule for hierarchical evidence. *Artificial Intelligence Journal* 33(3) 271-298

Shum, S. (1991). Cognitive Dimensions of Design Rationale. in Diaper, D. & N. V. Hammond (Eds.) *People and Computers VI.* New York: Cambridge University Press

Stefik, M., G. Foster, D. Bobrow, K. Kahn, S. Lanning & L. Suchman (1987). Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of ACM* 30(1) pp. 32-40

Steinberg, L. & T. M. Mitchell (1985). The redesign system: A knowledge-based approach to VLSI CAD. *IEEE Design Test* 2 45-54

Swartout, W. (1983). XPLAIN: a system for creating and explaining expert consulting programs. *Artificial Intelligence Journal* 21(3) pp. 285-325

Toulmin, S. (1958). *The Uses of Argument.* Cambridge Univ. Press: Cambridge, England

von Winterfeldt, D. & W. Edwards (1986). *Decision Analysis and Behavioral Research.* Cambridge University Press: New York

Webber, B. & N. Nilsson, Eds. (1981). *Readings in Artificial Intelligence.* Tioga Publishing Company Palo Alto, CA.

Wile, D. S. (1983). Program developments: formal explanations of implementations. *Communications of ACM* 26(11) pp. 902-911

Winston, P. (1980). Learning and reasoning by analogy. *Communications of ACM* 23(12) pp. 689-703

Winston, P. (1982). Learning new principles from precedents and exercises. *Artificial Intelligence Journal* 19(3) pp. 321-350

Winston, P. (1982). Learning new principles from precedents and exercises. *Artificial Intelligence Journal* 19(3) pp. 321-350

Yakemovic, K. C. B. & J. Conklin (1990). Report on a development project use of an issue-based information system. *Proceedings of Computer Supported Cooperative Work* LA, CA. pp. 105-118

Zloof, M. M. (1978). Query-by-Example:a database language. *IBM Systems Journal* 16(4) 324-343