1991

What's in Design Rationale?

Jintae Lee

Kum-Yew Lai

What's in Design Rationale?

Abstract
A few representations have been used for capturing design rationale. To understand their scope and adequacy, we need to know how to evaluate them. In this article, we develop a framework for evaluating the expressive adequacy of design rationale representations. This framework is built by progressively differentiating the elements of design rationale that, when made explicit, support an increasing number of the design tasks. Using this framework, we present and assess DRL (Decision Representation Language), a language for representing rationales that we believe is the most expressive of the existing representations. We also use the framework to assess the expressiveness of other design rationale representations and compare them to DRL. We conclude by pointing out the need for articulating other dimensions along which to evaluate design rationale representations.

Disciplines
Finance and Financial Management | Social and Behavioral Sciences

The International Center for Research on the Management of Technology

A Comparative Analysis of Design Rationale Representations

Jintae Lee
Kum-Yew Lai

March 1992 **WP # 84-92**

Sloan WP# 3295 & 3405

CCS TR# 121

A revised and condensed version of this report appears in the special issue of Human-Computer Interaction on design rationale, v.6(3-4), pp.251-280.

Sloan School of Management
Massachusetts Institute of Technology
38 Memorial Drive, E56-390
Cambridge, MA 02139-4307

A revised and condensed version of this report appears in the special issue of
Human-Computer Interaction on design rationale, v.6(3-4), pp. 251-280

A Comparative Analysis of Design Rationale Representations

Jintae Lee and Kum-Yew Lai

Center for Coordination Science
and MIT Artificial Intelligence Laboratory

ABSTRACT

A few representations have been used for capturing design rationale. It is important to know in what ways they are adequate or limited so that we know how to improve them. In this paper, we develop a framework for evaluating design rationale representations based on a set of generic design tasks. We build the framework by progressively differentiating the elements of design rationale that, when made explicit, support an increasing number of the design tasks. With this framework, we evaluate the expressiveness of the existing representations. We also present a language, DRL, that we believe is the most expressive of the existing representations without being too complex for human users. We also discuss the limitations of DRL as open problems for further research.

1. INTRODUCTION

As the articles in this issue point out, an explicit representation of design rationale can bring many benefits. Such a representation can lead to a better understanding of the issues involved [McCall 1987; Yakemovic & Conklin 1990], of the design space [MacLean et al. 1989], and of the principles underlying human-computer interaction [Carroll & Campbell 1989]. It can also provide a basis for learning, justification, and computational support [Fischer et al. 1989; Lee 1990a]. The extent to which we can actually reap these benefits, however, would depend largely on the language we use for representing design rationale. If, for example, design rationale were represented in free text, the benefits we obtain from it would not be different from what we already get from the notes on paper that we take in design meetings. Also, the

Authors' present address: Jintae Lee and Kum-Yew Lai, Center for Coordination Science, Massachusetts Institute of Technology, E40-140, 1 Amherst Street, Cambridge, MA 02139.
Electronic mail addresses: jin@ai.mit.edu and kumyew@eagle.mit.edu.

CONTENTS

kinds of computational support that we can provide depends on what a representation makes explicit and how formal the representation is. A few systems have been built and actually used to capture design rationale or arguments [Kunz & Rittel 1970; McCall 1987; Conklin & Begeman 1988; Fischer et al. 1989; Lee 1990a, 1990b; McCall 1990], and most of them used representations based on the earlier studies of design activity [Kunz & Rittel 1970] or of argumentation [Toulmin 1958]. However, there is no systematic attempt to justify the choice of these representations or discuss the rationale for using them.

This paper is motivated by the following questions: How adequate are the existing representations? Do they allow us to easily represent what we want to represent? In general, how do we evaluate a language for representing design rationale? This paper is an attempt to answer these questions by identifying the elements of design rationale that could be made explicit and by exploring the consequences of making them explicit. Laying out these elements provides a framework for placing the existing meanings of design rationale in perspective, as well as providing a framework for evaluating a representation language for design rationale, as we hope to show in this paper.

We proceed in the following way. First, we identify the tasks that we might want to support using a design rationale representation. Throughout the paper, we will use these tasks as a reference against which we would evaluate the representations that we discuss. In Section 3, we characterize design rationale by presenting progressively richer models. We start with a simple model of design rationale, where an artifact is associated with a body of reasons for the choice of that artifact. We then elaborate this simple model by incrementally differentiating and making explicit what is implicit in the body of reasons. As we do so, we discuss what each resulting model allows us to do. These models of design rationale provide a framework in which to define the scope of a representation and its adequacy within its scope.[1] Using this framework, we evaluate the existing representations in Section 4. In Section 5, we present a language, called DRL, which we believe is more expressive than most of the existing representations and overcomes many of their limitations. As we describe DRL, we also discuss its current limitations, which we present as open problems for future research.

2. WHAT DO WE WANT TO DO WITH DESIGN RATIONALE?

To evaluate a representation, we need to know what tasks it is designed to support. The tasks that a design rationale representation can or should support can be described in many ways at different levels of abstractions.[2] For example, [Mostow 1985] lists the following tasks: documentation, understanding, debugging, verification, analysis, explanation, modification, and automation. [Fischer et al. 1991] points out that documenting design rationale can support maintenance and redesign of an artifact, reuse of the design knowledge, and critical reflection during design process. [MacLean et al. 1991] list two major benefits from design rationale representation: aid to reasoning and aid to communication. The tasks of achieving these benefits are elaborated further in terms of subtasks such as enabling the designers to envisage the available design alternatives in a more structured way, with the arguments for and against them.

Another way of characterizing the tasks is to list the questions that we often need to answer to accomplish the general tasks mentioned above in the design process. To the extent that we want our design rationale representation to help answer these questions, answering these

---

1 In this paper, we use the terms, model and representation, in the following way. Model is an algebraic concept, and representation a linguistic one. Since the same structure can be described in many ways, a model can have several representations. Therefore, when we want to discuss a structure independent of a particular way of describing it, we use the term model. On the other hand, we use the term representation to refer to a particular notation for describing the structure.

2 There are many good discussions of the benefits or tasks for which a design ratioanle can be used. Those not discussed in this paper include [McCall 1987 ;Yakemovic & Conklin1990].

questions become the tasks that the representation should support. The following is a set of representative questions that we gathered from our experiments with design rationale [Lee 1991a], from walking through examples [Lewis et al. 1991], and from creating scenarios [Carroll & Rosson 1990].

- What is the status of the current design?
- What did we discuss last week and what do we need to do today?
- What are the alternative designs and what are their pros and cons?
- What are the two most favorable alternatives so far?
- Sun Microsystems just released their X/NeWs server. How would the release change our evaluations?
- What if we do not consider portability?
- Why is portability important anyway?
- What are the issues that depend on this issue?
- What are the unresolved issues? What are we currently doing about them?
- What's the consequences of doing away with this part?
- How did other people deal with this problem? What can we learn from the past design cases?

The above list of the questions is by no means complete as there are many possible paths that we did not walk through and many scenarios that we did not construct. We also left out those questions which, though important, do not seem to be the job of design rationale to answer (e.g. How can we compute the total cost of this design?) Nevertheless, the questions in the list provide a useful framework for assessing the expressiveness of the different representations. When we discuss the limited or increased expressiveness of a given representation, we refer to those questions that can or cannot be answered as a result. If our task includes answering a question which is not represented in the list, then we can always evaluate the representations by asking whether they would support answering the question and if not, what additional objects, attributes, or relations would have to be made explicit.

We want to emphasize further that we are assessing only the expressiveness of the existing representations of design rationale. That it is desirable to answer the questions in the list does not mean that any representation for design rationale should support the answering of these questions. Each representation must weigh the costs and benefits involved in tradeoffs among three general dimensions: expressiveness, human usability, and computational tractability (Figure 1). These tradeoffs should in turn be motivated by the tasks that are intended to be

accomplished using the representation. In short, we are not dictating what any existing representation should or should not have. However, we do hope that the analysis presented in this paper would make the architects of a representation for design rationale more conscious of what their language can or cannot express and why.

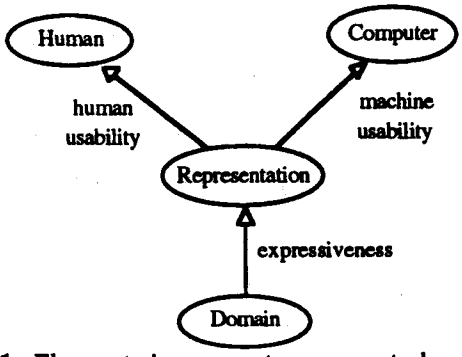


Figure 1. Elements in computer supported activities.

To be sure, we cannot separate our concern with expressiveness entirely from other concerns such as human usability or computational tractability. For example, if a language is meant to be used by people to capture design rationales, but if it is too complex for people to manage, then there is not much point in evaluating its expressiveness. Whether any of the representations that we discuss falls into that category is an empirical question. All the languages discussed here have been actually used by people, but that is no guarantee that they will all succeed at their "industrial strength" use [Conklin & Yakemovic 1991]. Nevertheless, we believe it would be difficult to evaluate tradeoffs among the three dimensions without calibrating individual dimensions such as expressiveness (cf. [Brachman & Levesque 1984] on tradeoffs between expressiveness and computational tractability for general knowledge representation).

3. WHAT'S IN DESIGN RATIONALE?

What is design rationale? Design rationale for an artifact has at least three different meanings in its current usage: a historical record of the reasons for the choice of an artifact [Yakemovic & Conklin 1990], a set of psychological claims embodied by an artifact [Carroll & Rosson 1990], and a description of the design space [MacLean et al. 1989].[3]

---

3 The representation used in [Yakemovic & Conklin1990] describes logical as well as historical aspects of design rationale, as we discuss further later. For example, the support relation has both the logical aspect and the historical aspect because . We associate [Yakemovic & Conklin 1990], as well as [Lee 1990a; Potts & Bruns 1988] with the historical record because one of their goal *is* to capture and document the actual process of design.

Design rationale often means the historical record of the analysis that led to the choice of the particular artifact or the feature in question. To illustrate, let us take as an example a particular feature of the MacIntosh operating system, namely the placement of all the window commands in the global menubar at the top of the screen.[4] By a window command, we mean a command specific to a window, e.g. SAVE is a window command that saves the contents of the window. A design rationale for this feature in the sense of historical record would be something like: "The issue of where to put the window commands was raised by Mark on January 20. Kevin proposed the idea of incorporating them into the global menu bar at the top of the screen and pointed out that it saves screen space (e.g., as opposed to putting the commands on each window, as in the Star environment). Julie objected because it requires a long mouse travel from the currently active window in executing a command. But, we decided to have the global menu bar anyway because people generally agreed that that the advantage, together with others such as more efficient implementation, outweigh the objection." We can provide more structure to this historical record, as we discuss in the rest of the paper. Such structure is usually designed to make explicit the logical structure (e.g., an argument supports a proposal) and/or the historical structure (e.g., a proposal replaces another proposal).

Another meaning of design rationale is the set of psychological claims embodied by an artifact in the sense of [Carroll & Kellogg 1989], i.e. "the claims that would have to be true if the artifact is to be successful" or "the claims about the psychological consequences for the user" [Carroll & Rosson 1990]. These claims are different from the historical record; the claims need not be present in the historical record; even if they were, they would have to be extracted from the record and formulated in a testable form. For example, the design rationale in this sense would be something like: "The global menu makes the environment easier to use because it reduces screen clutter." or "Dimming the irrelevant items in the global menu makes it easier to learn about the commands."

The third meaning of design rationale is one used by [MacLean et al. 1989], namely how a given artifact is located in the space of possible design alternatives: what are the other possible alternatives, how are these alternatives related, what are the tradeoffs among them? In our MacIntosh example, the design rationale in this sense would be some description of the logically possible alternatives for placing window commands, how they are related, and what the tradeoffs are. It is often difficult to provide such a description in a systematic way, but an

---

[4] Since the unit of an artifact is often ambiguous (e.g., MacIntosh, MacIntosh OS, MacIntosh window system, MacIntosh menubar, or the position of the menubar), we will use the term artifact in the general sense to mean any feature, usually a small one, for which there is design rationale.

example is found in [Card et al. 1990; McKinlay et al. 1990], which provides a vocabulary of the primitives and a set of composition operators for describing the design space of possible input devices. This meaning of design rationale seems different from the first meaning in its emphasis on design rationale not being a record but a construction, and from the second in its emphasis not on a particular artifact but on the relation among possible alternatives.

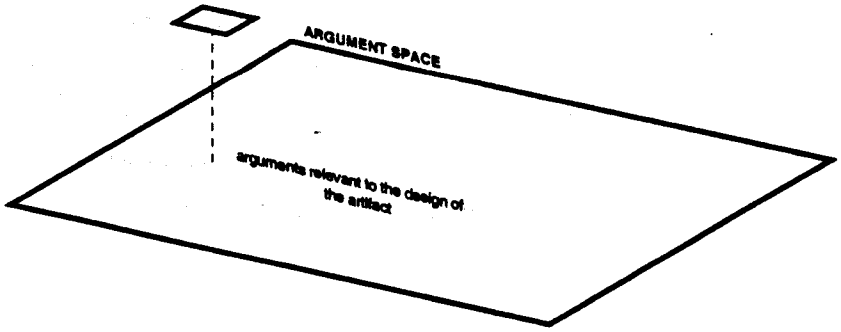
Models of Design Rationale

We now develop a series of progressively richer models of design rationale, which provide a framework in which we can place the three different meanings of design rationale. The first two meanings are discussed immediately below. The third meaning of design rationale, as a possibility space, is discussed shortly after.

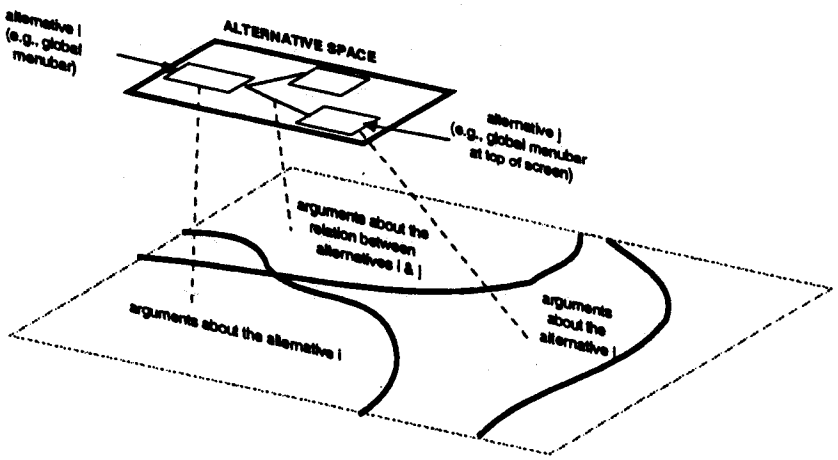
Design rationale in the most general sense is an explanation of why an artifact is designed the way it is. So in our first model of design rationale, an artifact is associated with a body of reasons as shown in Fig. 2a.

There are different kinds of reasons that we can give for an artifact. The reasons can be historical or logical, roughly corresponding to the meanings of design rationale, respectively, as a historical record and as a set of claims embodied by an artifact.[5] The record of the process that led to the choice of an artifact tells us one kind of reason why that artifact was chosen. The free text example about the window commands above is an example. If we wanted a logical justification, we would have to extract it from the record; but at least such a record tells us the historical circumstances and sequence that led to the design, and provides a basis from which to infer the logical reasons. On the other hand, we can represent the logical reasons directly, i.e. the reasons justifying the choice of an artifact no matter how or in what order they were articulated. The set of claims embodied by an artifact is an example because these claims would justify the design of the artifact. These claims are logical also in the sense that the context in which they are true have to be made explicit. For example, a claim should not say "The global menu is better because it leads to smaller implementation" if it really means "The

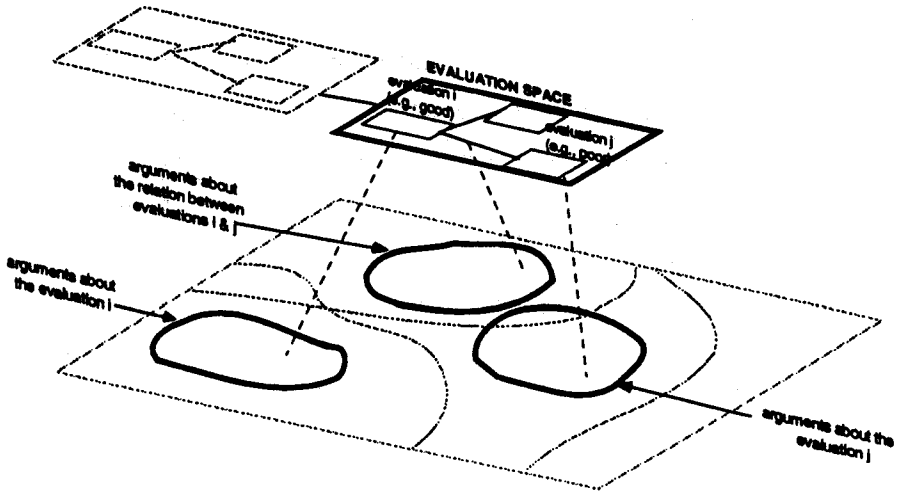
5 We believe that the distinction between historical and logical reasons breaks if we push it too far because a purely historical record per se does not really give us a reason. It would give us a reason only to the extent that we can extract some logical structure out of it. Nevertheless, we believe that the distinction is useful for the purpose of evaluating representations because for a given representation we would like to know what it makes explicit and what we have to infer from it.



(a) MODEL 1: An artifact is associated with a body of all the arguments relevant to the design of the artifact.

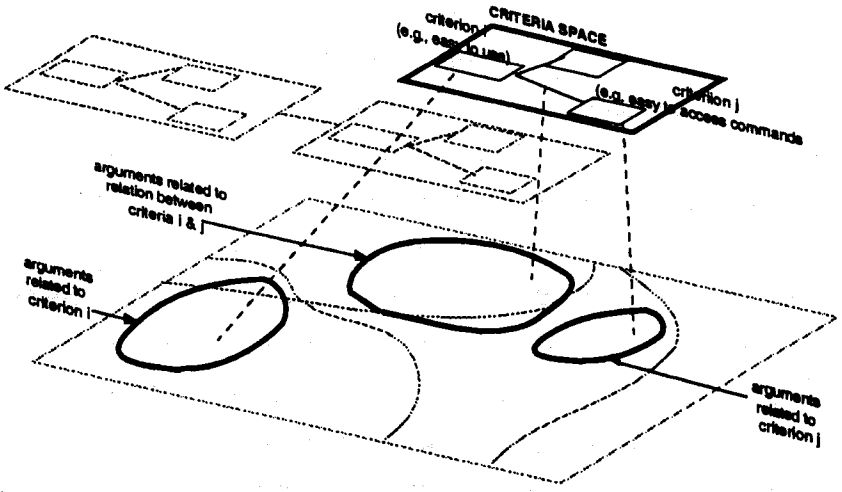


(b) MODEL 2: Alternatives and their relations are made explicit and the arguments about individual alternatives can be differentiated.

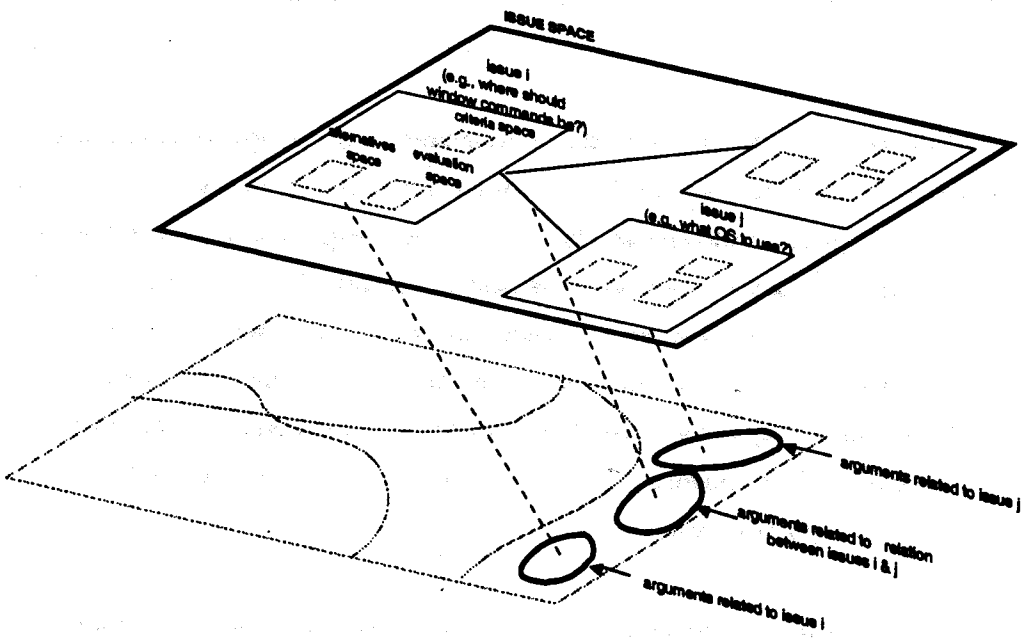


(c) MODEL 3: Evaluation measures used and their relations are made explicit and the arguments about them can be differentiated.

Figure 2 (a)-(c). Progressively more differentiated models of design 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 2 (d)-(e). Progressively more differentiated models of design rationale (continued from previous page).

global menu is better in the context of the MacIntosh because it leads to smaller implementation. This is important in a system like the MacIntosh which has a small memory." Extracting these logical reasons is not an easy task; once identified, however, they provide the advantages of being testable and general.

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 that we gave earlier as an example of a historical record. If we were to make the historical relations more explicit, we can differentiate further by making explicit such roles and relations such as: *Initiator, Second-Motioner, Initiates,* and *Replaces.* An example that is not historical is the representation used by [Carroll & Rosson 1991] for describing the claims "embodied" in an artifact. In this representation, the claims themselves are represented in natural language, but the claims are grouped by the questions they answer to: what can I do, how does that work, and how do I do this? We can also imagine a representation where the logical support relations can be made more explicit by providing such constructs as *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 historical record of the various arguments relevant for the design of the artifact or logical arguments underlying the design.

There is much we can do with our first model of design rationale. A representation based on this model can help us answer the question, "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? Can we learn from the past design cases? Carroll and Rosson [Carroll & Rosson 1991] provide a good example. They report in detail how their representation of design rationale, mentioned above, suggested many issues for redesigning an artifact (the View Matcher in Smalltalk). They discuss how these issues can be couched as a design hypothesis, which can be tested and compiled to form, in the long run, "a contextualized science out of practice" of human computer interaction.

Our first model, however, does not help very much with the other questions, though we should qualify this statement immediately. Saying that it does not help much is not to say that we cannot answer these questions. Of course, if the user works hard enough, and as long as the representation based on the model has enough information captured, even in the form of natural language free text, we can answer these questions. So the real issue is how much the model itself help us answer these questions either by helping us see the structure better or by enabling us to define computational services that help us answer the questions. 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 1991].

Our second model (Fig. 2b) differs from the first by making multiple alternatives and their relations explicit. Design involves formulating several alternatives, comparing them, and merging them, as many of the questions in our list indicate. In our first model, only a single alternative is made explicit at a given time, and the multiple alternatives are present only implicitly in the argument space. Our second model make these 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. comparing alternatives, displaying the alternatives that specialize this 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 and the knowledge used in evaluating them are useful in other contexts, for example, when situational constraints change. We use the term, Alternative Space, to refer to this set of multiple alternatives and their relations.

These relations among the alternatives can also 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 these different alternatives and their relations, we say that the alternative space is within the scope of the representation. gIBIS, for example, seems to include the alternative space within its scope because one of its goals is "to capture alternative resolutions (including those which are later rejected), [and] trade-off analysis among these alternatives" [Conklin & Begeman 1988]. The constructs in gIBIS for representing the alternative space consist of: *Position*, with which we can describe multiple alternatives, and the specialization relation among the *Positions*.

By now, we have an alternative space connected to the argument space, as shown in Fig. 2b. 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. 2b.

Once the alternative space is represented, we can imagine how we can make a system that help us answer some of the questions in our list. To answer "What are the alternative designs and what are their pros and cons?," we can associate an argument space with each of the alternatives through the links such as *Supports* or *Objects-To*, as in gIBIS. To answer "Why do we even consider this alternative, and how is it related to the one that we discussed last week?" we need to some historical relations, such as *Replaces*, among the alternatives. With the representation of the alternative space, we can imagine how we can make a system help us answer some of the questions posed in Section 2. To answer "What are the alternative designs and what are their pros and cons?," we can associate an argument space with each of the alternatives through the links such as *Supports* or *Objects-To*, as in gIBIS. To answer "Why do we even consider this alternative, and how is it related to the one that we discussed last week?" we need to use some historical relations (e.g. *Replaces*) or structural relations (e.g. *Is-A-Part-Of*), among the alternatives.

However, once we make explicit multiple alternatives, we need to articulate more carefully what the argument space is about (Fig. 2b). 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 the evaluation space (Fig. 2c), where the evaluation status are made explicit and 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 when we leave it for 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 that automatically propagates and merges them 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, a calculus for combining them, and a model specifying what they mean. Even in the case where these actions are left to human, for example. if the human user is expected to combine these values to produce a higher level summary measure, then 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. That is, as shown in Fig. 2c, these different kinds of arguments can be differentiated in the argument space. With the representation of the evaluation space, we can now answer questions such as: "What are the two most favorable alternatives so far?" and "Sun Microsystems just released their X/NeWs server. How would the release change our evaluations?" We can also explain how an evaluation was made by pointing to the arguments in the argument space behind the artifact in question, and explaining how this particular evaluation measure is derived or computed from them or 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 "We do not need to duplicate menu items" is a pro-argument for the alternative "Global Menu at the top of the Screen" *because of* the goal of reducing screen clutter, 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, then we can do appropriate things to all the arguments that presuppose this goal (for example, making these arguments less important). Knowing how this criterion is related to others (e.g. "Reducing Screen Clutter" is a way of achieving "Easy to Use"), also allows us to assign proper importance to this criterion or change its importance, when the related criteria changes. We use the term criteria space to refer to these criteria and their relations. As Fig. 2d 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. 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.

With the criteria space represented, we can now see how the system might be able to help us answer the questions such as: what if we do not consider portability? or why is portability important anyway? The answer might be "If we give up the goal of 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 from a representation if the representation makes explicit the relation between evaluations, criteria, and arguments. Of course, representation of the criteria is not sufficient for answering these questions. It is not obvious how these questions can be answered even if some parts of the criteria space are represented explicitly. However, the explicit representation of the criteria space seems a necessary condition if we are to answer 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 some examples of such operations later in Section 5.

So far, we have identified and discussed the structure of a *single* decision underlying an artifact, namely which of the alternative designs 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 we currently doing about them? 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. A unit in this issue space is, therefore, a single decision that has as its internal structure the other spaces, as shown in Fig. 2e. Once we have an issue as an explicit element, we can associate the attributes such as "Status" and "Actions Taken" with issues and answer questions such as "What are the unresolved issues and what are we currently doing about them?" Representing the dependency relation among the issues will allow us to answer the question "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? Can we learn from the past design cases? 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 Section 5, we discuss computational operations that help us answer these additional questions by exploiting the structure of the five spaces.

4. EXISTING REPRESENTATIONS

There are only a handful of representations that have been used for representing design rationale [Toulmin 1958; Kunz & Rittel 1970; Marshall 1987; McCall 1987; Conklin & Begeman 1988; Potts & Bruns 1988; Lee 1990a; MacLean et al. 1991]. Most, though not all, of these representations are heavily influenced by the IBIS structure for representing issues [Kunz & Rittel 1970] or by Toulmin's model of argumentation [Toulmin 1958]. [MacLean et al. 89; MacLean 91] also propose a representation based on their experiences with constructing design rationales. In the next three subsections (4.1-4.3), we discuss these three representations in detail by defining their scope and evaluating them within the framework outlined above. In the next subsection (4.4), we discuss a few other studies that bear on design rationale representation. In Section 5, we present a language, called DRL, which we believe overcomes many of the limitations of the existing languages that we discuss in this section.

Some qualifications are in order before we proceed. Our intention is not to criticize the existing representations, but only to evaluate them *as* a design rationale representation language. Even then, our evaluation is mainly with respect to their expressiveness. We would like to emphasize again that the adequacy of a representation can be evaluated only with respect to a set of tasks. One representation may be much more expressive than another; as a result we may be able to do more with it. However, if what it enables is not in the set of desirable tasks, or if it enables those only by sacrificing other more important constraints, then the additional expressiveness is not worth what it gives us. What we discuss below is only what the existing representations allow or do not allow us to do; We do not intend to make value judgments about whether the representations should or should not do. Some of these languages were designed with different goals, which in turn determine the tradeoffs adopted in the designs. Therefore, the following discussion should not be construed as a criticism of these representations, but only as an articulation of their scope and their adequacy as design rationale representation languages with respect to expressiveness.

We also want to make clear that whenever we say that a representation cannot express some information, it does not mean that people cannot infer that information from the

representation. Take a natural language representation of design rationale that we gave as an example in the beginning of the paper. If we keep a detailed enough record of what happened, or even a video recording of the whole design process, we can always retrieve the information that has been ever 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 see the structure better or that makes it amenable to computational manipulation.

4.1. IBIS (Issue Based Information System)

The IBIS structure was originally developed in [Kunz & Rittel 1970] for the purpose of representing designers' argumentation activities. Since then, some variations of it have been used by a few systems for representing design rationale. One variation is the representation used by gIBIS [Conklin & Begeman 1988], "a hypertext tool for exploratory policy discussion." Since gIBIS is most well-known [Conklin & Yakemovic 1991] and can be regarded as a modern incarnation of the original IBIS , we use gIBIS as the context for discussing the IBIS representation.[6] Other variations include PHI (Procedural Hierarchy of Issues) [McCall 1987] and the one used by [Potts & Bruns 1988] for the rationale module in their representation. We discuss them briefly in Section 4.4.

The goal of gIBIS is to capture "the design problems, alternative resolutions, tradeoff analysis among these alternatives, and the tentative and firm commitments that were made in the process of the decision making process". Figure 3 shows the objects and relations that form the language of gIBIS and Figure 4 shows an example representation. In gIBIS, one raises an *Issue* such as where to put the window commands. *Positions* are created to *Responds-to* the issue (e.g., "in the global menubar at the top of the *screen.*", and "Commands at the top of *each window.*") *Arguments* can be created to *Support* or *Object-to* a Position. For example, the argument "Don't need to duplicate commands for each window" supports the first position, and the argument "Requires long mouse travel." objects to it. Also, an *Issue* can be related to other objects as shown in the figure. The gIBIS model extends the original IBIS model by introducing: the generalize/specialize relation among *Arguments* as well as among *Positions*, an additional

---

6 We believe that the distinction between historical and logical reasons breaks if we push it too far because a purely historical record per se does not really give us a reason. It would give us a reason only to the extent that we can extract some logical structure out of it. Nevertheless, we believe that the distinction is useful for the purpose of evaluating representations because for a given representation we would like to know what it makes explicit and what we have to infer from it.

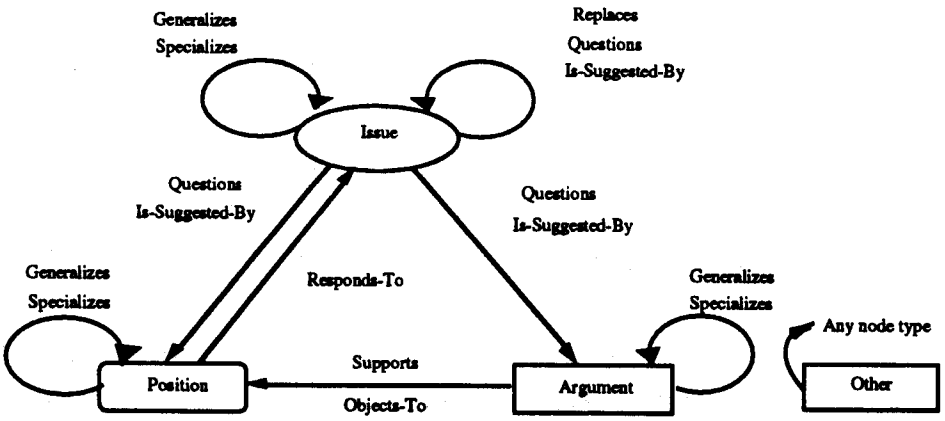


Figure 3. The gIBIS vocabulary.

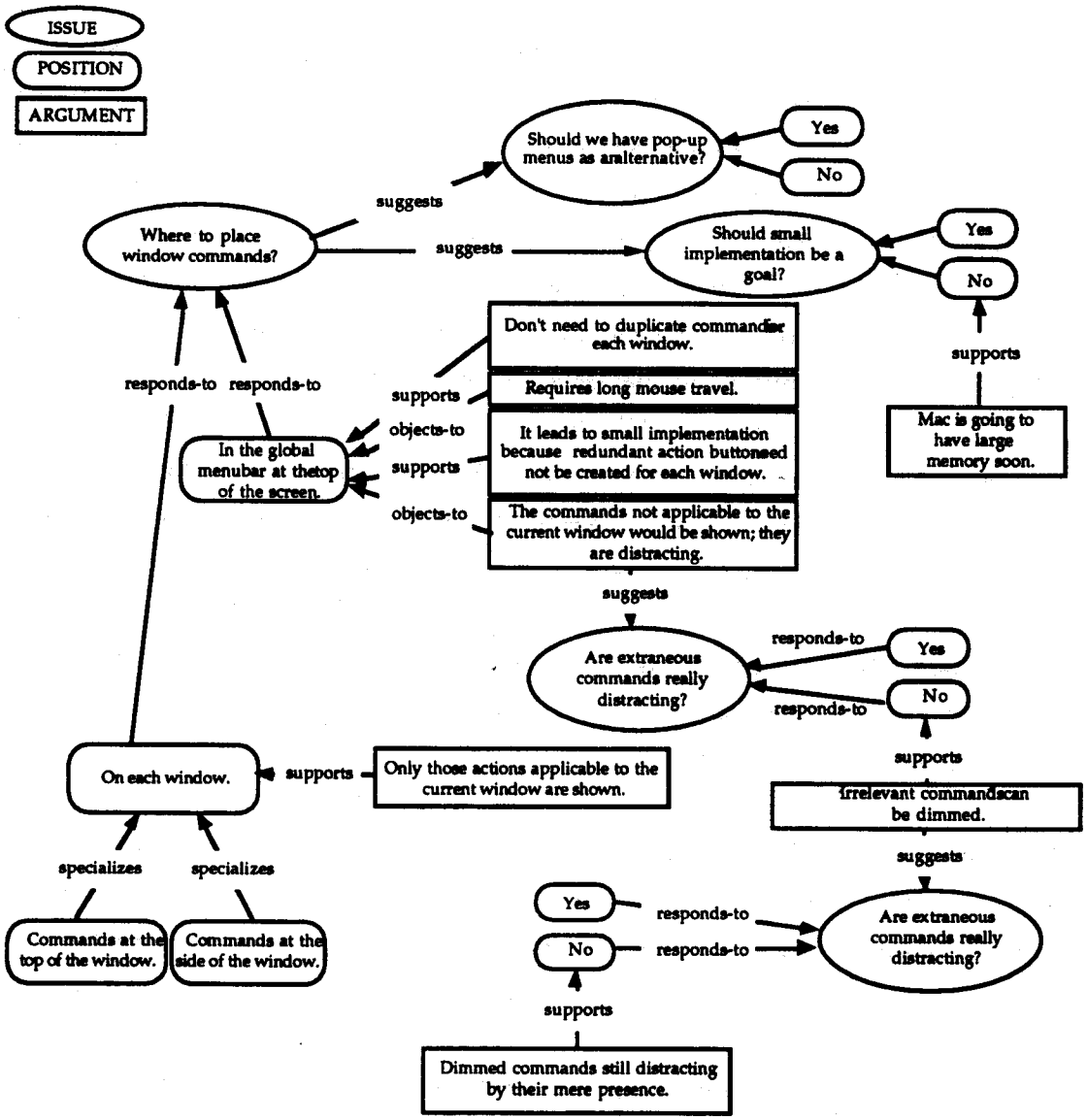


Figure 4. An example in IBIS.

Other type which serve as an "escape" mechanism for representing things not representable with the available constructs, and an *External* type for external objects such as documents or sketches. In the following, our discussion refers to the structure common to both IBIS and gIBIS except when we discuss the generalize/specialize relation for *Position* and *Argument*.

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, as we discuss below.

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*).[7] Thus, gIBIS can answer the question "What are the other alternatives being considered?" but not the question "How are these alternatives related?" unless they happened to be related via the *Specializes* relation.

The unit of the *argument space* in gIBIS is *Argument*. An *Argument* (e.g. "The commands not applicable to the current window would be shown and distracting.") can support or object to a *Position* (e.g. "In the global menubar at the top of the screen"), indicating a reason why the position is or is not a good one. There is no construct, however, for qualifying an argument. For example, we cannot indicate that the existing *Argument*, "the global menu reduces the screen clutter", is valid only if the *Argument* "the global menu containing all possible action items does not become so big itself" is valid.

The relation among the arguments is again limited to the *Specializes* relation. This limitation implies that an *Argument*cannot *Support* or *Object-to* another *Argument* directly. For example, the *Argument* "Irrelevant menus can be dimmed" is an argument objecting to the original *Argument*, "The commands not applicable to the current window would be shown and distracting." To express this relationship among *Arguments* in gIBIS, we have to create an *Issue* about whether the original *Argument* is right or not, create *Positions* "yes" and "no", and then

---

7 In the original IBIS, response to an argument was said to have been allowed in the form of "counter-argument." However, I was not able to find any discussion of it yet.

argue about these *Positions*.[8] A possible criticism of this representation is that it leads to the proliferation of *Issues*. For example, in order to just point out the factual inaccuracy of a claim, we would have to create three new objects: a new *Issue*, a "No" *Position*, and an *Argument* supporting the "No" *Position*. However, we might be able to reduce the complexity with an interface that hides the intermediate details. A more serious limitation of this representation is that it does not help us answer questions such as "Show me all the claims that respond to this claim." We might try to answer these questions by following the *Is-Suggested-By* link that connects the original claim and the issues that contain the responding claims. However, the *Is-Suggested-By* link is too general for that purpose because it does not allow us to distinguish the issues that contain responding arguments from those that do not.

In gIBIS we cannot argue about relational claims. For example, there is no way of saying that we agree with A and B, but not that A *Supports* B because a link in gIBIS is not an object that you can argue about. To illustrate with another example, suppose we want to argue about whether portability should be a criterion for evaluating the positions for a given issue. The only way we can do so in IBIS is by creating a new *Issue*, say "Should we consider portability as a criteria?", and suggest possible answers in the form of positions, say "Yes, we should" or "No, we should not". Then, we can argue about these positions. We can also relate this new issue to the original issue by an *Is-Suggested-By* relation. However, this representation again does not make explicit the relation between the new issue and the criteria being questioned. If we make a relation an explicit object, then we can argue directly about the relation, *Is-a-Criteria-for*, between the object portability and the original issue.

Being able to argue about relational claims is important for other reasons as well. People often want to say that they may agree with A and B but not with the claim that A supports B. For example, one may agree that the global menubar is a bad idea and that seeing irrelevant commands is distracting but not that the second claim supports the first. That is, one may believe that the global menubar is a bad idea but not because it distracts the user by showing irrelevant commands; he may point out that the global menubar does not have to show the irrelevant commands. The user may want to make the distinction clear because they do not want to be construed as denying either of the claims alone. The distinction is even more crucial when the representation is to be used to provide computational services. Mixing denial of node

---

8 Another alternative is to show the first argument as supporting the position that the second objects to, instead of objecting to the second argument directly because an argument cannot respond to another argument in gIBIS. This representation does not make explicit the relation between the two arguments. Probably for that reason, gIBIS does not encourage this representation (Conklin, personal communication)

with denial of link just is not right semantically. If you define a computational service based on this type of careless semantics, such as evaluation management that propagates and merges evaluations, what you get would not make much sense.

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," but beyond that it is not clear what more sophisticated evaluation measures can be defined on the gIBIS structure. We could assign uncertainty measures to the arguments, and try to derive some ordinal ranking among the positions based on supporting or objecting arguments. However, it is difficult to imagine what the calculus would be like.

The criteria space is beyond the scope of gIBIS. gIBIS provides no vocabulary for describing the criteria we use in evaluating the alternatives. This lack of explicit representation of criteria in gIBIS 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 mutually exclusiveness, among the criteria. Again, we could indirectly represent these relations by creating additional *Issues*. For example, Fig. 4 shows an *Issue* ("Should small implementation be a goal?") related to our original *Issue* via an *Is-Suggested-by* link. But, again, this representation does not make explicit the relation between this issue and the criterion in question. It will be difficult for people to see this relationship; it will be even more difficult for a machine. Lack of explicit representation of the criteria entails other limitations. For example, when goals 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. As we will see in the next section, explicit representation of goals can provide modular representation of arguments, multiple viewpoints, and a basis for relevance matching.

In gIBIS, the unit of the issue space 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*. Figure 4 shows that the original *Issue* about the placement of window commands suggests other *Issues*: whether the command pop-up menu should be considered as an alternative and whether we should have small implementation as a goal. An *Issue* can also *Question* a *Position* or an *Argument;* an *Issue* can be *Suggested-by* a *Position* or by an *Argument*. These relations are quite important in describing a more global relationship among the issues. 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.

4.2. Toulmin's Model of Argumentation

Stephen Toulmin, a philosopher, proposed a model of argument in 1958. Since then, the model has been adopted for many purposes, including the computational representation of arguments [Birnbaum et al. 1980; Lowe 1986; Marshall 1987; Streitz et al. 1989]. Figure 6 shows an example Toulmin representation of an argument.[9]

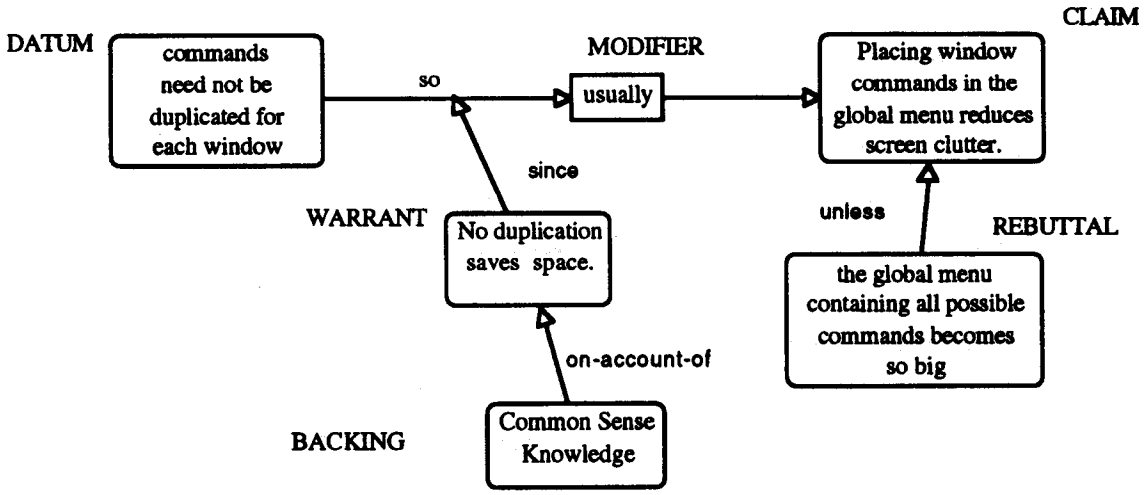


Figure 5. An example of an argument in Toulmin's representation.

A *Claim* is the main assertion being made (e.g., "Placing the window commands in the global menu reduces the screen clutter.") We support the *Claim* by producing a *Datum*, "Commands need not be duplicated for each window."[10] We justify the leap from the *Datum* to the *Claim* by producing a *Warrant*, e.g. "If you need not duplicate things, that saves space." A *Warrant* is usually an inference rule or a principle that licenses the inference from *Datum* to *Claim*. A *Warrant* is not often absolute; so if we want to express how strongly this *Warrant* supports the inference from the *Datum* to the *Claim*, we use a *Qualifier*. In our example, the *Qualifier* is

---

9 We will use the term Toulmin's representation rather than Toulmin's model because we are interested in the model to the extent it has been or can be used as a representation.

10 As pointed out in [Newman & Marshall 90], Toulmin wants the datum to be a singular fact, such as something that you can point to, because we think he still wants to retain the syllogistic form, where one gets the conclusion by having as the minor premise, a singular fact. So he seems still under the bound of syllogism, at least at the time of his writing, despite his rebellion against it. The datum of our example is a more general statement. But we use this example first because the distinction itself between a singular fact and a universal statement has been called into question in philosophy and because in design it would be unusual to support a claim always with a singular fact.

"usually", which qualifies the strength of the inference. We indicate the excepting circumstances where the *Warrant* fails through *Rebuttal* (e.g., "The object to be shared might have to be larger than the sum of these objects locally existing.") If a *Warrant* is called into question, then we appeal to a *Backing*, i.e., a body of knowledge that tell us why in general we should accept the warrant. In our example, we appeal to our commonsense knowledge. Other types of backing might be physical laws, historical facts, or legal precedents.

The scope of the Toulmin structure as a design rationale representation is limited to what we called the argument space. The goal of Toulmin structure is to make explicit how a claim is supported. There is no attempt to represent alternatives, goals, any evaluation measure, or how they are related. As such, we evaluate Toulmin's representation only as a representation module for the argument space, which might interface with representations for the other elements of design rationale. Even then, we should keep in mind that the original goal of Toulmin was to delineate the logical support structure of an argument, not necessarily to provide a representation to be used dynamically for capturing design rationale.

As a representation for the argument space of design rationale, Toulmin's has many limitations. For example, we can only deny a *Claim* by supporting its negation. Suppose you want to deny that the global menu reduces the screen clutter because the global menu has to contain all possible command items (let us assume that this is before pull-down menus were invented). The only way to do so is by creating a *Claim* "the global menu does not reduce the screen clutter" and presents "the global menu has to contain all the menu items" as the *Datum* for this new *Claim*. Whether denying "A reduces B" means the same as supporting "A does not reduce B", it is certainly an awkward way of denying a *Claim*; and in any case, without some construct that indicates that the two *Claims* are the negation of each other, the denial relationship would not be explicit at all. Even with such construct, such awkward representation would slow down human comprehension or machine computation. Some people [Birnbaum et al. 1980] have solved this problem by explicitly naming the original link between *Datum* and *Claim* as *Supports* and introducing another link called *Attacks*. So Toulmin's representation should be extended at least this much to accommodate denying claims.

It is also not easy to qualify a claim using Toulmin's representation. We can qualify a *Warrant* using *Qualifier* and *Rebuttal*, but if we want to qualify a *Claim* or a *Datum*, there is no corresponding construct. We could of course build into the *Claim* any qualification so that the *Claim* itself is of the form, "Global menu reduces screen clutter *provided* that the global menu is not too big," but then we might as well use a natural language as our representation. The

purpose of making these components distinct is to allow us to see the relations more clearly, to formally manipulate these components and relations, and to incrementally construct pieces of the argument.

Toulmin's representation also suffers from building into object types context-dependent roles, such as warrant or backing. The difference between warrant and backing is not in the nature of the object itself but only in the fact that backing is something that supports warrant. The same object can be a *Warrant* or a *Backing* or even *Claim* or *Datum*, depending on the context. So either we assign a type to an object based on the inherent properties of the object and indicate its context-dependent role in some other way, as we show in Section 5, or an object type should dynamically change depending on the context. Most of the Toulmin-based representations [Marshall 1987; Streitz et al. 1989; Newman & Marshall 1990] take the latter approach. For example, if we want to support a *Datum* object, we first change its role from *Datum* to *Claim*, then instantiate the five-component schema, and fill it in. However, then it is not clear why we need a separate type called *Backing*. If we want to support or deny a *Warrant*, why not change its role to *Claim*, instantiate the now four-components (that is, the Toulmin structure without *Backing*) schema, and use a *Datum* to back the *Claim*? It is not clear what is special about *Warrant* that we need a separate type for its support when we do not need a separate type to support other things such as a *Datum*.

We understand that Toulmin's reason for introducing *Backing* as a separate type is to provide a category for the kind of knowledge that is rarely called into question, such as the written law, common sense knowledge, or physical laws. We agree that for an argument to be resolved, there must be something that the involved parties can agree and appeal to. However, we believe that it is misleading to represent such knowledge as a distinct component of a rigid structure such as Toulmin's. For one thing, the five-component schema leads people to believe that an argument has to make explicit all these components to be complete. This belief often forces people to unnecessarily represent the obvious; for example many warrants, when explicitly represented, are often as obvious as the one in our example, "No duplication saves space". Also often in the process, people are led to argue about whether something is a warrant or backing, rather than about the substance of the argument. Conversely, the closed nature of the schema leads people to believe that if an argument has these five components, it is complete. But, having five components is arbitrary, because there might have to be multiple *Data*, multiple *Warrants*, or chain of *Warrants* before we get to something that we agree on. We believe that it is better to define the representation in an open-ended way that allows as many objects to be created as needed without necessarily specifying the boundary of an argument and without

requiring the obvious to be represented unless it needs to be, e.g., unless it is called into question or argued about.

There are numerous other limitations with Toulmin's representation, as [Newman & Marshall 1990] point out from their experiences of using Toulmin's for representing legal arguments. For example, they had to make *Rebuttal* have at least four additional meanings to represent the different types of objections, each corresponding to what is being objected to. We believe that these different types of objections should explicitly indicate what they object to, and that we can do so in a graceful way, as we hope to show in Section 5.

4.3. QOC (Question, Option, and Criteria)

QOC is a representation proposed by [MacLean et al. 1989, 1991] for "constructing" design rationale. 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. This emphasis makes QOC different from, say gIBIS, whose goal at least includes capturing the rationale as it unfolds.

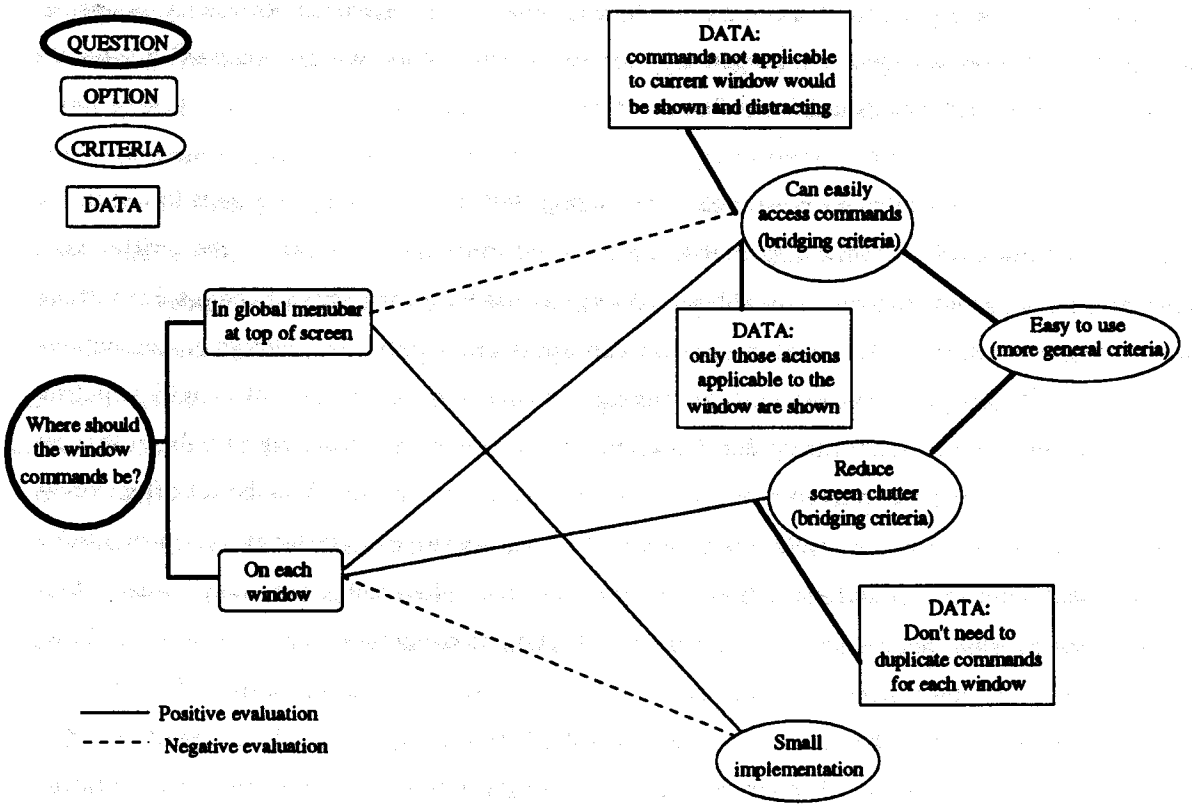


Figure 6. A partial representation in QOC of the menubar example.

The major constructs of QOC are straightforward and map clearly to the framework proposed in this paper. Figure 6 shows an example represented in QOC.[11] 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, QOC can represent the criteria space with *Criteria*. A *Criteria* (e.g. Reduce Screen Clutter) 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. Easy to use). The units of evaluation space are links labelled with "+" and "-," 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. "The mouse is a Fitt's Law Device") or to an accepted *Theory* (e.g. "Fitt's Law"). 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. [MacLean et al. 1991] provide an illuminating discussion of the other forms of justifications for design, such as various forms of dependencies and metaphors, no specific constructs are discussed for representing them.

QOC as we understand it has a number of 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, it is not clear how an argument such as "Irrelevant commands can be dimmed" should be treated given that it is neither a piece of empirical data 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* seems to treated as a fixed type, rather than a role that a criterion plays with respect to another criterion. However, building a role into a fixed type results in unnecessary inflexibility. That is, being a *Bridging Criterion* is not a property inherent in the object itself but is a relation that the object has to another criterion. As such, one should not have to classify a given object as a Bridging Criterion but instead indicate its role through the relation it has with another criterion. 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,

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

how we can differentiate those arguments that support the overall evaluation from those that deny or qualify it. Suppose that the empirical *Data*, "Commands can be found in fixed positions,"produces the positive evaluation of the *Option*, "The global menubar at top of the screen," with respect to the *Criterion* "Can easily access commands." Suppose, however, that there is another argument (e.g. "It requires longer mouse travel.") against the option with respect to the same criterion. We can represent the design rationale in one of two ways: (1) create two evaluation links, one labelled + and the other -, between the same *Option/Criterion* pair, or (2) create only one link labelled with whatever the net effect of these pro and con arguments is. The latter option has the disadvantage of not differentiating explicitly how each datum contributes to 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*.

4.4. Other Representations

In this section, we discuss a few other representations, explicitly for design rationale or otherwise, whose scope includes the aspects we discussed above.

PHI (Procedural Hierarchy of Issues) [McCall 1987] overcomes many of the limitations with gIBIS 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, an *Issue*, A, is a child of another *Issue*, B, if A "serves" B -- that is, if 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 -- for example, a mechanical system is a subanswer of a system. 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. This way, the expressive power for the argument space is extended over that in gIBIS.

On the other hand, the expressiveness of PHI is limited in the following aspects. It is not clear how to express the specialization of issues in PHI because the natural representation of specialization, namely the hierarchical relation, has been preempted by the serve relation. By the same token, it is not clear how to represent the serve relation in the answer space because the hierarchical relation in the answer space is the specialization relation. That is,

how do you represent an *Answer* that serves another *Answer* in the sense that adopting the first (e.g. Use a large screen) will facilitate adopting the second (e.g. Reduce Screen Clutter)? PHI also inherits many of the limitations of IBIS with regard to the argument space: we cannot qualify a claim, cannot make the criteria explicit, and cannot argue about the relational claims, for example, that A *Supports* B. Again, PHI does not make the criteria space explicit.

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, that is 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. The model extends the IBIS model to represent the derivation history of an artifact design including a software 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, 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. On the other hand, this representation still does not represent the logical relationship among the alternatives. Relying on the IBIS structure for representing its rationale, it shares the problems of IBIS such as: the criteria used for justification are not explicit, an argument cannot respond to another, and we cannot argue about relational claims.

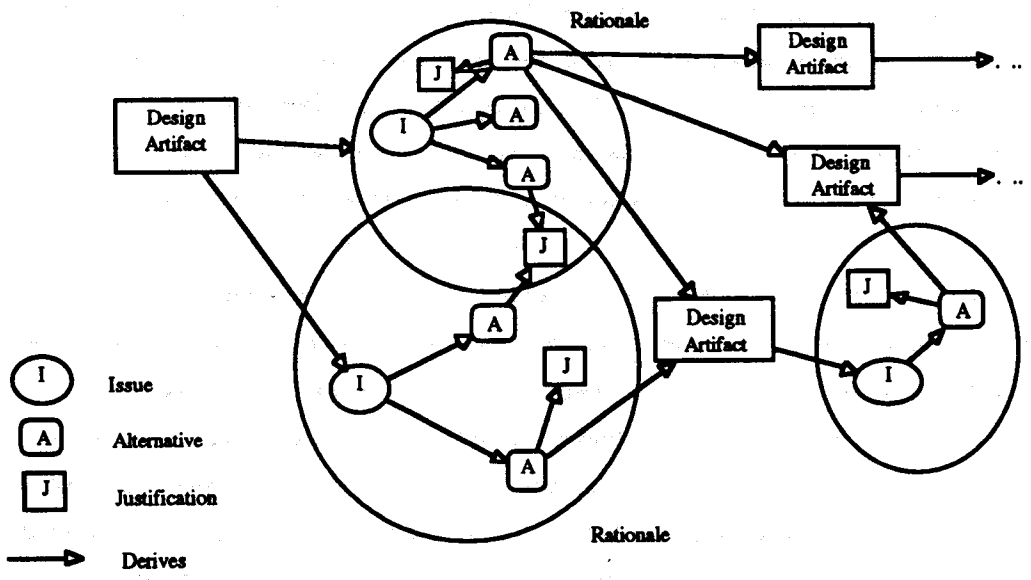


Figure 7. The Potts and Bruns representation schema.

[Marshall 1987] uses a matrix to integrate the various elements of design rationale. The matrix shows the evaluation of each alternative with respect to each goal in each of the cells. The representation used for each of the four spaces is quite simple. The alternative space is simply a set of alternatives; there is no representation of how they are derived or related. The same is true about the criteria except that the sub-criteria relations can be suggested by nested matrices. The evaluation space is simply an ordinal scale consisting of symbols such as "++," "+," "–," "—," and "0;" but Marshall's is one of the few studies that address the evaluation space explicitly by discussing how the measures could be combined and what they mean. The representation of the argument space for the evaluations is also simple, but allows us to support, deny, or qualify an argument. Each evaluation has an associated claim structure, where a claim can have evidence and assumptions. Although each claim is simple in itself, the claims become powerful when they are integrated through the matrix structure. For example, it makes the tradeoff relationship among the alternative clearer. Furthermore, Marshall suggests the use of the matrix to inversely infer the goals from the decisions an agent made. However, the representation is limited in many ways: there is no way to question or argue about evidence or assumptions, nor about the claim that they are evidence or assumptions, nor about criteria or alternatives.

ARL [Smolensky et al. 1988] is probably the most fine grained of the existing languages for representing the argument space. It provides a rich vocabulary for describing the various elements of an argument and the relations among them. It also allows the user to extend the vocabulary by defining a new term out of the existing terms. The primary goal of ARL,

however, is to represent argument structures typically found in academic papers and to serve as an underlying structure for aiding "reasoned discourse", especially writing. The structure of the arguments in design deliberation seems sufficiently different from the structure of the arguments in academic papers that ARL would not be a good design rationale representation language. Also, ARL does not provide any construct for representing goals, alternatives, or evaluations. However, ARL raises interesting questions about the domain of arguments: Do arguments have different structures in different contexts? Or are the differences only a matter of how fine-grained are the distinctions we are willing to make in the different contexts? Studying these questions may help us better understand the nature of arguments in general.

5. DRL (DECISION REPRESENTATION LANGUAGE)

5.1. Introduction

DRL [Lee 1990a] is a language that we have developed for representing and managing the qualitative elements of decision making: for example, the alternatives being considered, their current evaluations, the arguments responsible for producing these evaluations, and the criteria used for the evaluations. We call *decision rationale* the representation of these qualitative elements, and we call *a decision rationale management system* a system that provides an environment for capturing decision rationale and computational services using it. Decision rationale in our sense is a subset of *design rationale* in that it does not represent some important aspects of design rationale. For example, a design rationale may include the deliberations about how to *generate* the alternative designs. The scope of DRL, at least for now, does not include the representation of such deliberations.[12] The exact relation between decision rationale and design rationale, however, has yet to be articulated. Nevertheless, we believe that DRL is the most expressive language that has been used for representing design rationale and that it overcomes many of the limitations in the existing languages in a way that is still simple enough for the user. In this section, we evaluate DRL as a design rationale representation language, and point out its strengths and limitations as such.

---

12 That is not to say that DRL does not help us generate alternative designs. It does in a couple of ways. DRL allows people to argue about the existing alternative designs, thus helping them to see more clearly their strengths and the limitations. It also helps people retrieve the past decisions that contain useful alternative designs which are still useful for the current decision or those that can be so adapted. Furthermore, DRL can represent the relationship between the existing alternative and the new alternative that may have been derived from it. However, being able to generate a new design alternative is still different from representing the rationales for how it was generated

5.2. Description of DRL

Figure 8 shows the objects and the relations that form the vocabulary of DRL. The argument types for a relation are shown inside the parenthesis following the name of the relation. All *DRL Relations*, for example, are subclasses of *Claim*, as we explain below. Figure 9 shows the relations between them graphically. Figure 10 shows an example decision rationale represented in DRL. We describe the basic features of DRL briefly, and discuss in detail how they allow us to represent the five spaces of design rationale. One should note that the following is not a description of the way that the user would use DRL. The actual interface is briefly described later.

A qualification is in order before we proceed further. DRL is an extensible language, and we have a developed a method, called Dependency Analysis, that can be used to systematically add additional constructs for a given set of tasks. Therefore, the constructs of DRL presented below are what we believe are essential for the generic task of decision making. For a given set of tasks, these constructs would be specialized to more task-specific constructs using dependency analysis. For example, if a task involves looking at the history of how the requirements evolved over time, then the relations among goals such as *Replaces*, *Elaborates*, and *Merges* would be important to represent explicitly. Unfortunately, we cannot discuss here, for want of space, the dependency analysis method that allows us to identify which additional constructs to provide and specify their semantics precisely. We hope to describe it in another paper soon.[13]

13 Very briefly, however, dependency analysis works as follows. Given a task, we generate all of its subtasks recursively to form a task hierarchy. From the leaves of the hierarchy, we identify the objects and the attributes that we need. For all possible pairs of the attributes, we can systematically generate a set of tables. Each of these tables corresponds to the influence that a particular type of change in the first attribute can exert on the other attribute. A subset of the cells in a table correspond to a relation, and the type of change signified by the cells in the set gives semantics to the relation.

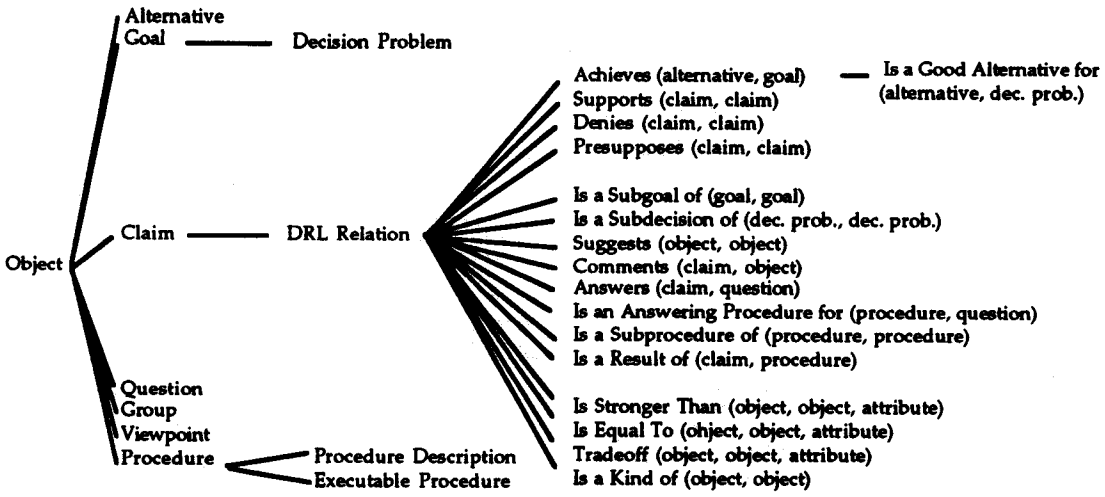


Figure 8. The DRL vocabulary.

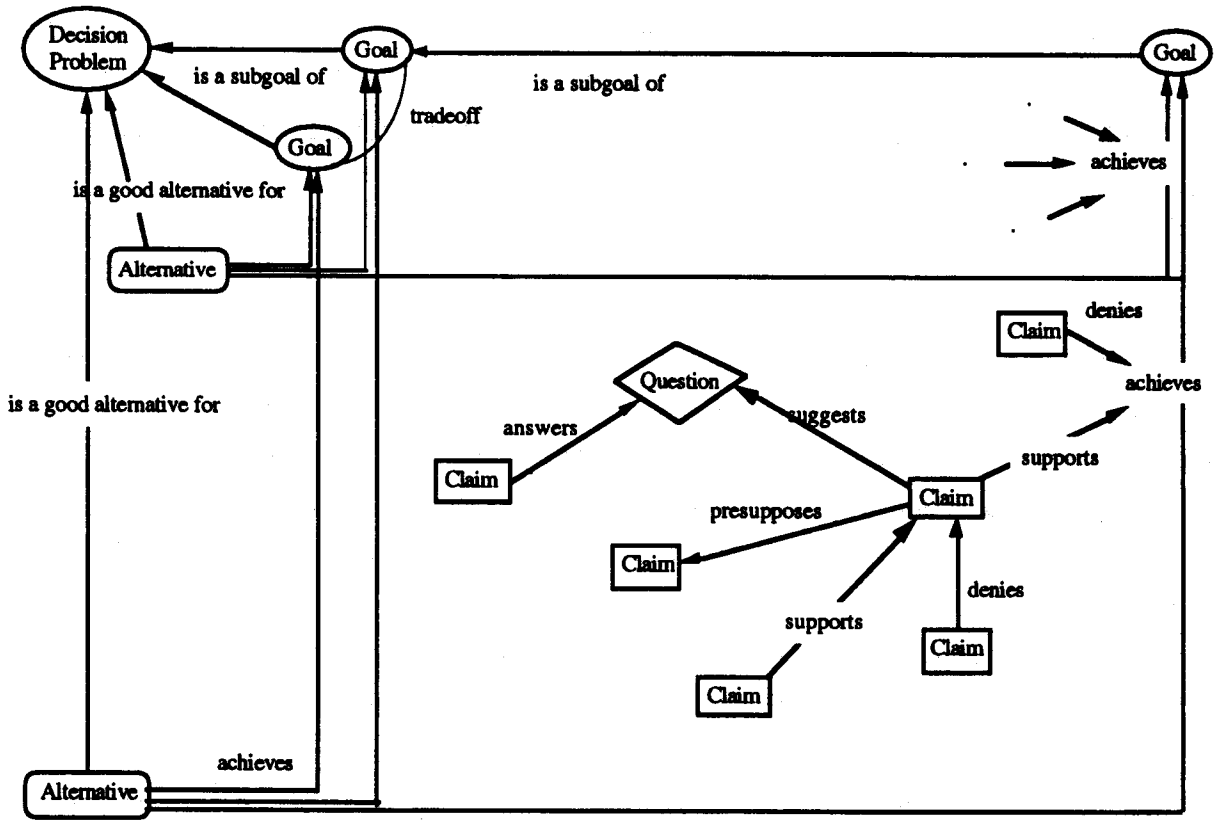


Figure 9. The structure of a DRL decision graph.

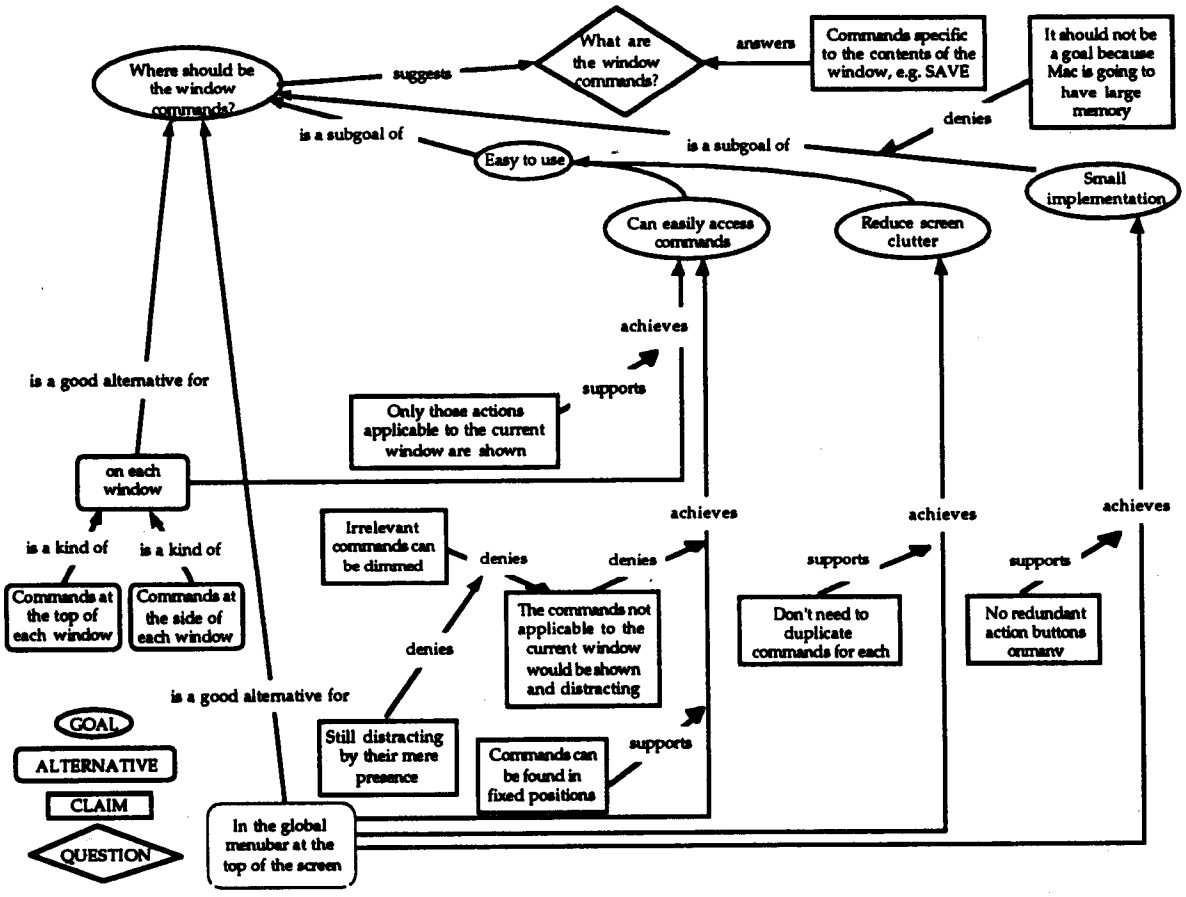


Figure 10. An example decision graph.

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. "global menu at the top of the screen." A *Goal* represents a desirable state or property used for comparing the alternatives: e.g. "Minimize screen clutter". A *Goal* is elaborated further in terms of its subgoals: e.g. "Easy to use" is elaborated to mean "Can easily access command items" and "Minimize screen clutter". Every relation in DRL is a subclass of *Claim*, as shown in Fig. 8. For example, the rightmost *Achieves* link in Fig. 10 represents the *Claim* that the *Alternative*, "the global menu at the top of the screen," achieves the *Goal* "Reduce screen clutter."

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*. Each *Claim* has the following attributes: "Evaluation," "Plausibility," and "Degree." The evaluation of a *Claim*,

represented by the value of its "Evaluation" attribute, is a function of both of its "Plausibility" and "Degree" attribute values. The "Plausibility" of a *Claim* tells us how probable the claim is true, and the "Degree" of a *Claim* tells us to what extent it is true. For example, the "Degree" of the *Achieves* link between the *Alternative* and the *Goal* tells us to what degree that the alternative achieves the goal in question. The overall evaluation of an alternative is represented by the "Degree" 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 degree is a function of the degrees of the *Achieves* claims that link the *Alternative* to the different *Goals*. Not all of the three attributes have to be used for the evaluation. For example, we might require that a *Claim* be entered only if its "Plausibility" is above a certain threshold, and ignore the "Plausibility" once the *Claim* has been entered. In that case, we can do away with the "Plausibility" attribute, and the "Evaluation" and the "Degree" attributes become synonymous.

There are other auxiliary objects in DRL. A *Group* object groups a number 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 other objects in DRL such as *Question*, *Procedure*, and *Viewpoint* represent somewhat auxiliary aspects of decision making such as the questions raised and the procedures used for answering the questions. We discuss more details of DRL in [Lee 1990a].

DRL has been implemented in a system called SIBYL, which runs on top of Object Lens [Lai et al. 1988]. Although the above description of DRL may seem complex, the actual user interface provided by SIBYL for using DRL is quite simple and SIBYL has been used for real group decision tasks such as designing a workplace layout. For example, SIBYL makes it easy to create objects like a *Decision Problem*, *Goals*, and *Alternatives* by providing context-sensitive menus and template editors. Once a *Decision Problem* and some of its *Goals* and *Alternatives* are specified, SIBYL displays them in a matrix such as shown in Fig. 11. By mouse-clicking on a cell of the matrix, the user gets the menu of all the actions that can be performed on the selected object. For example, by mouse-clicking on a *Goal*, the user can get a menu containing action items such as creating a new subgoal, or displaying a goal tree showing how this *Goal* is related to other *Goals*.

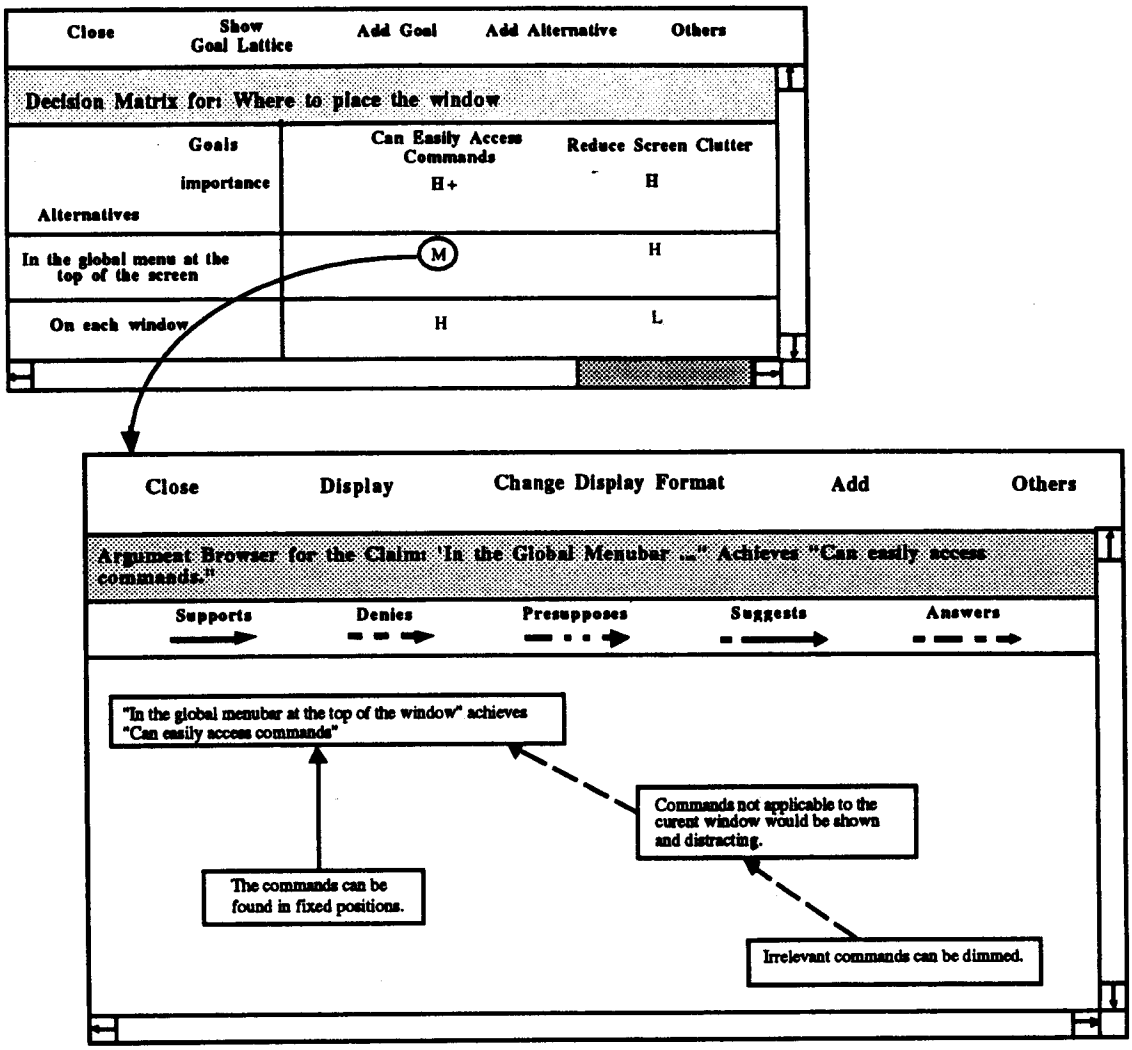


Figure 11. Clicking the mouse on a cell (containing an *Achieves* claim) in the decision matrix in SIBYL produces an argument browser with all the claims that are related to the *Achieves* claim.

Figure 11 also shows an argument browser displayed when the user chooses the action, "Display Arguments," from a pop-up menu that appears when one of the evaluation cells is selected. The argument browser shows, in a network format, all the *Claims* that provide reasons for the evaluation, i.e., all the claims related to the surrogate claim, namely that a given alternative achieves a given goal, that the system automatically generates to be argued about. By mouse-clicking on an object in the argument browser, the user gets a pop-up menu of all the operations that can be performed on the object: such as "Add a Supporting Claim," "Add a Denying Claim," and "Add a Qualifying Claim." When the user chooses one of these actions, the template editor containing a new *Claim* object appears, and the new object is added to the argument browser with an appropriate link to the object chosen. The user interface that SIBYL

provides for using DRL is described in more detail in [Lee 1990b]. Using the decision rationale represented in DRL, the computer can provide many services such as managing the dependencies among claims, propagating and merging the plausibilities automatically, providing multiple viewpoints, and retrieving useful knowledge from past decisions. We discuss these services in [Lai et al. 1988].

5.3. Evaluation of DRL as a Design Rationale Language

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. Also, if we need to make the distinction, say, between facts and assumptions, we can do so simply or by specializing a claim or by using nominal categories like "facts" and "assumptions" as values for the "Plausibility" attribute in different *Claims*. We can in fact 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 the nominal categories such as facts or assumptions. We discuss different plausibility measures when discussing the evaluation space.

A *Claim* can be *Supported, Denied,* or *Presupposed* by another *Claim*. These relations among the *Claims* allow us to respond to a *Claim* directly without, as in IBIS, having to indirectly respond to the *Position* that responds to the second *Claim*. For example, the *Claim*, "Irrelevant commands can be dimmed," directly denies the *Claim* "The commands not applicable to the current window would be shown and distracting," rather than having to be formulated as a *Claim* for the *Alternative* in question. These direct relations among the *Claims* allow us to see the logical and the dynamic structure of the argument more easily. All DRL relations are special types of *Claims*. For example, when we link a *Claim* 1 to *Claim* 2 through a *Supports* relation, we are making the claim that *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; i.e. people can support, deny, or qualify them. For

example, "Commands not applicable to the current window would be shown and distracting." is denied by "Irrelevant commands can be dimmed". That the first *Claim* is denied by the second itself is a relational *Claim*, which is then denied by "Dimmed commands are still distracting by their mere presence."

Unlike Toulmin's argument structure, the roles of a *Claim* are represented by its relations to other *Claims*, not by changing its types such as to *Datum* or to *Warrant*. Figure 12 shows the DRL equivalent of the Toulmin schema. *Qualifier* is represented as the value of the "Plausibility" attribute of the *Supports* relational claim between the two *Claims* at the top. *Rebuttal* is not shown because it is ambiguous what a rebuttal is denying. This representation of roles in terms of relations allows us to see the global picture of the argument structure without being confined to the micro-structure defined by the Toulmin schema and to expand the structure where and when we need to. We can also deny exactly and directly the thing that we want to deny rather than having to ambiguously overload the meaning of rebuttal.

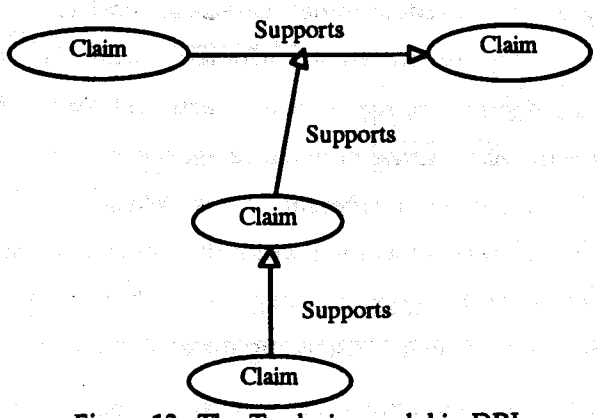


Figure 12. The Toulmin model in DRL.

The Criteria Space

DRL represents the criteria space fairly well. 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, and because we want to convey the richer relationship 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*, as shown in Fig. 13.

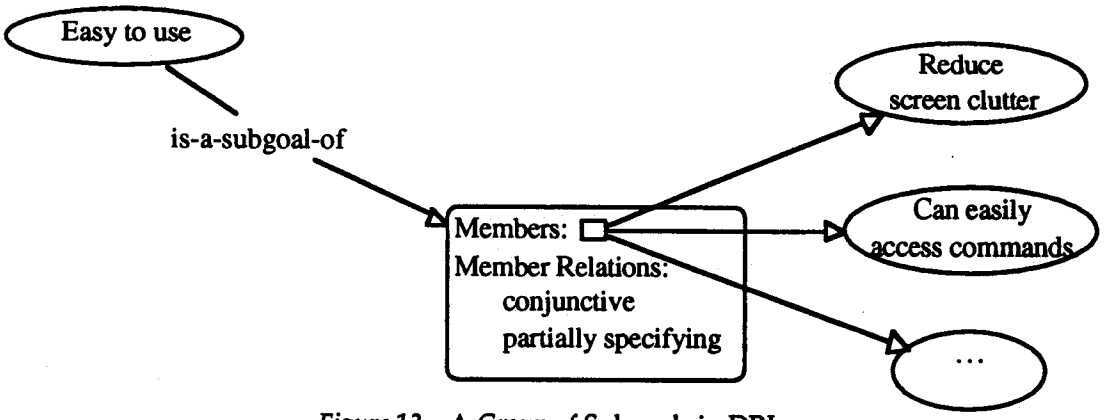


Figure 13. A *Group* of Subgoals in DRL

Decision Problem represents the goal of choosing the best alternative. All the other goals for the decision problem are subgoals of the decision problem in the sense that they elaborate what it means to choose the best alternative. For example, the *Goal* "Easy to use" is a subgoal of the *Decision Problem* of our example if we interpret it to mean "Choose the alternative that has the property 'Easy to use.'" In other words, satisfying this goal facilitates the achievement of the goal of choosing the best alternative.[14]

Since 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 small implementation should be a goal at all. In Fig. 10, there is an argument, "It should not be a goal because Mac is going to have large memory soon," denying that it is a subgoal of the decision problem, i.e., small implementation is not a desirable property that should be used to compare alternatives. The record of these *Claims* and their relations represents the argument space for the goal space. Elsewhere [Lai et al. 1988], we discuss how this representation of *Goals* in DRL allows us to create multiple viewpoints, and extract from past decisions knowledge useful to the current decision.

---

14 The precise semantics of the model underlying DRL is more complicated and fully explained in [Lee, 1991a]. Roughly, for a given decision problem of the form, what is the best alternative for X, its underlying interpretation is "the goal of choosing the best alternative for X". Its subgoal of the form, G, is strictly speaking "the goal of choosing the alternative that satisfy G". An alternative of the form, A (e.g. the global menubar at top of the screen), is to be interpreted as "Choosing the alternative A". It is in this sense that a decision problem is the parent of the other goals and that an alternative achieves a goal. This nicety, though important for computational purposes, can be ignored by human users.

The Alternative Space

DRL represents only parts of the alternative space well. DRL can represent alternatives and the specialization relation among them through the *Is-a-Kind-of* relation. Thus, we can say that "Commands at the top of the window" is a special case of the alternative, "On each window". However, design alternatives may be related in much more complex ways than through the specialization relation. An *Alternative* can be related to another *Alternative* via, for example, the following relations: *Elaborates, Simplifies,* or *Is-the Next-Version-of. Alternatives* can be related in a more complex way, for example, in the context of a design space. These relations are beyond the current expressive power 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, 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, we can say that "commands on each window" is an alternative only if "the window system allows the attachment of menu windows to the main window" by linking the two claims with a *Presupposes* relation. One can of course object to this *Claim,* in turn, by pointing out another way of implementing the window commands at the top of the window.

The Evaluation Space

In DRL, each *Claim* has the attributes: "Evaluation," "Plausibility," and "Degree." The "Evaluation" attribute tells us how important the claim is, and its value is a function of both "Plausibility" (how likely the claim is true) and "Degree" (to what extent the claim is true). The overall evaluation of an alternative is represented as the "Evaluation" attribute value of the relational claim, *Is-a-Good-Alternative-for,* between the *Alternative* and the *Decision Problem.* This value represents the extent to which the alternative satisfies the overall goal. This value, in turn, is a function of the evaluations of the *Achieves* relations that link the *Alternative* to the subgoals of the *Decision Problem* (i.e., the extent to which the alternative satisfies the subgoals). 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.

DRL does not commit to a particular measure of evaluation. The user of DRL can use nominal categories, numeric measures, or whatever they devise for evaluation. However, such evaluation measures should come with the algorithm for propagating and merging them to produce evaluation measures at a higher level. For example, if we want to use probability as the measure of plausibility, then we should also know how the probability of the two *Claims* – "the alternative 'In the global menubar...' achieves the goal 'Can easily access commands'" and "The alternative 'In the global menubar ...' achieves the goal 'Reduce screen clutter'" – combine over the subgoal relations to produce the probability of the *Claim* "The alternative 'In the global menubar ... ' achieves the goal 'Easy to use'," given that we also know how these subgoals are related among themselves and to the parent goal. We might try to work out such an algorithm based on Bayes' theorem, for example.

However, as discussed in Section 2, the exact algorithm is important only to the extent that the user can trust the algorithm. That is, if the algorithm is based on many assumptions that the user feels are seriously violated, then the exactness of the algorithm does not contribute much. We might as well concentrate on how to support people for making these judgments. DRL takes this philosophy and tries to help by modularizing and helping to make explicit the relationship that need to be considered for these judgments.

The Issue Space

In DRL, the unit of the issue space is a decision problem. A *Decision Problem* corresponds to an *Issue* of gIBIS and a *Question* in QOC. A Decision Problem *Is-A-SubDecision-Of* another Decision Problem if a decision for the first requires a decision for the second. For example, deciding where to place the window commands might require deciding what the window layout algorithm is, e.g. tiling or overlapping. A Decision Problem IS-A-KIND-OF another Decision Problem if the first decision problem is a special case of the second: e.g., "Where to place the emacs window commands?" is a special case of "Where to place the window commands?" Of course, we can relate the decision problems through the generic relations such as *Is a kind of and Is a part of.* We are sure that there are many other possible relations. For example, the *Replaces* relation in IBIS seems important for describing the dynamic aspect of the issue space. Also, the *Is-Suggested-By* relation in IBIS is a vague but nevertheless useful as a catch-all relation to represent any relation that is not described by others. However, again, DRL is based on the philosophy that the vocabulary should be extended to tailor the task in hand and that it is better to provide a method for extending the vocabulary as needed rather than provide constructs that may not be useful in general.

5.4. Relation to Other Studies:

In this section, we relate DRL to the existing representations that we discussed in the previous section.[15]

The DRL structure helps us extract the claims embodied by an artifact in the sense of [Carroll & Kellogg 1989]. Although we have to be careful not to confuse the claims about the psychological consequences of an artifact for its user with the designer's intention or reasons for the artifact [Carroll & Rosson 1990], the designer's claims captured in the design process are often about the psychological consequences of the artifact. To that extent, making the designer's claims explicit help us to identify the psychological design rationale. For example, we can extract from the example in Fig. 10 the following claims embodied by the artifact of the MacIntosh menubar:

- The irrelevant commands in the global menubar is not distracting if dimmed.
- Placing the window commands in the global menubar facilitates ease of use because it would reduce the screen clutter by not duplicating the commands for each window
- Longer mouse travel required by placing the window commands in the global menu is not a serious detraction from the easy usability.

Some of these claims may be obvious, but even then they are worth stating explicitly because they can then become subject to tests or further elaboration. Especially, when we try to decontextualize claims, some obvious claims may not be obvious any more. For example, we might all agree that "Longer mouse travel required by placing the window commands in the global menu is not a serious detraction from the easy usability" but only if the screen size is small so that the extra distance to travel is at most small. DRL does not allow us to identify all the claims embodied by an artifact, only those that come up explicitly in the course of design. Furthermore, not all the claims that come up in the course of design are worth stating as the claims in Carroll's sense. the representation of design rationale in DRL through SIBYL help identify some of these claims, and can be viewed as an tool for extracting these claims.

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, gIBIS has

---

15 ArgNoter [Stefik et al. 87] is one of the earliest attempts to capture design rationale. Although DRL is designed to support many of the goals underlying Argnoter, we do not discuss ArgNoter here because no specific representation has been proposed for realizing its goals.

quite limited expressiveness, as we have discussed above: We cannot directly respond to an argument; we cannot represent or argue about the evaluation criteria used. We cannot indicate the relationship among alternatives or arguments. We cannot say that we disagree that A *Supports* B without disagreeing with A nor B. Most notably, the criteria used in the arguments evaluating the alternatives remain implicit. DRL can be viewed as extending gIBIS in several ways: the explicit representation of the criteria space, the richer representation of the argument space, provision of the infra-structure for defining evaluation measures, and the argument space not only for the evaluation but also for the alternative and the criteria spaces as well

The gIBIS structure has the advantage of being simple, and it is an empirical question what this simplicity buys or costs us. Its structure (*Issue, Position, Argument*) is general enough to represent most of the discussion, although it does not explicitly reveal many of the relations among the components of design rationale. If this simple structure makes people more likely to use the language, then it is certainly worth the while. It is better to be simple than not to be used at all. However, a simpler structure does not always enhance usability. People can be frustrated with not being able to say exactly what they want to say, e.g., that I agree with A and B but not that A objects to B. If a language with a finer structure allows people to say what they want to say and yet is natural enough for people to use, we would prefer such a language. From a computational point of view, more expressive structures would allow us to do more things with them. Also, we should remember that often it is not only the language itself but also the user interface that determines usability. Based on our limited experiences, we believe that the use of DRL in the context of SIBYL is not any more cumbersome than the use of the gIBIS structure. But it is an empirical question, and we plan to find out.

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 to us. *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. However, we pointed out earlier that it is not clear how to argue about these QOC constructs for the argument space. 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*. Whether we can argue about other links, for example, the relation among *Criteria* or the relation between *Data* and *Theory*, is not clear.

It seems that QOC is currently more a model rather than a fully-developed representation language. That is, it is an attempt to understand and categorize the elements of design rationale without providing yet a specific set of vocabulary for expressing them. It is of course a worthy endeavor and consistent with their warning against a premature commitment to a representation. Given the similarity 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.

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.

In [Potts & Bruns 1988], IBIS is used as the representation for the rationale component in their representation. The component is modular enough, however, that DRL can be viewed as an alternate representation for the module. [Fischer et al. 1989] also uses IBIS for its issue base, VIEWPOINTS. Again, DRL can be used as the alternative representation for the module, which we believe is more natural to interface with the other component, CRACK, because DRL is more expressive and can better support knowledge-base operations.

6. CONCLUSIONS

A large body of research in the last two decades or so points to the importance of choosing a right representation for a given task [Amarel 1968; Bobrow 1975; Winston 1984; Brachman & Levesque 1985]. The task of using and reusing design rationale is no different. The benefits we can get and how easily we can get them depends heavily on the representation we use. The choice of representation is especially important when a human is the user of the representation, as in design rationale capture, because a wrong representation can turn people away from the task altogether, attributing the failure and frustration to the task itself rather than the inadequacy of the representation used. People might conclude that capturing design rationale is not worth the trouble because it is so hard and because it does not provide enough rewards for

the efforts. But the real problem might be that the representation does not allow us to represent easily what we want to represent or in a way that can provide much benefit. Thus, it is important that we know how to evaluate a representation for a given task, in our case, for capturing design rationale.

In this paper, we made a step forward by characterizing the domain of design rationale, i.e., by identifying the kinds of elements that form the rationale as well as the relations that hold among them. Characterizing this domain is important because we then know what we can represent, what we have decided not to represent, and what are their consequences. It also helps us to map the different meanings of design rationale by associating them with the different parts or aggregates of the domain. In other words, it provides a framework for defining the scope and assessing the expressive adequacy of a representation. Using the framework, we defined the scope of the existing representations and discussed their adequacy. We have also presented a language, called DRL, which we believe is more expressive than most of the existing languages and overcomes many of their limitations in a way that is still natural to human users. However, that is a testable claim that we plan to investigate empirically by using DRL with many tasks by many users. We also discussed the limitations of DRL, which we hope will be the topics for future research by others as well as by us.

The step we made, however, is a small one and we have a long way to go before we fully understand the important issues in designing an ideal representation for representing design rationale. We provided a framework for evaluating a design rationale representation along one dimension -- its expressive power. Even then, expressiveness involves more than being able to represent the elements in the domain explicitly or not. There are many other characteristics, such as the ability to provide abstractions, that are important [Bobrow 1975]. We need to think about whether these characteristics matter much for the task we have in hand, and in what way they matter. Then, there are other dimensions to a representation than its expressiveness. In Section 2, we mentioned two other categories: human usability and computational tractability. We need to articulate the characteristics that make a representation more usable. We also need to identify the computational services that we can provide with design rationale so that we know what they require and any tradeoff between their requirements and other requirements such as human usability. There are some existing studies which address these problems [Lee 1989; Newman & Marshall 1990; Yakemovic & Conklin 1990; Conklin & Yakemovic 1991; Fischer et al. 1991; Lee 1991b; Shum 1991]. We need more such studies, more focus on the representation being used, and more systematic categorization of the results. We

believe that the benefits from explicit representation of design rationale would more than pay for the efforts that we put into such studies.

Acknowledgments. The whole project of articulating the domain arose from our frustration with not knowing exactly how to evaluate the existing representations and how to relate them to DRL. Tom Malone suggested that we define the scope of a representation by thinking about which components of a decision matrix it makes explicit. That insight triggered much of the analysis here. One of us (Jintae) would like to thank Frank Halasz for providing him with the great environment in which he wrote this paper. The comments from the anonymous reviewers, Jeff Conklin, Tom Moran, Randy Trigg, and Austin Henderson were valuable. The paper was influenced much by the comments from the members of ARG (Argumentation Reading Group): Danny Bobrow, Frank Halasz, Bill Janssen, Cathy Marshal, Susan Newman, Dan Russell, Russ Rogers, Mark Stefik, and Norbert Streitz. We also thank the members of the learning group at the MIT AI Lab, especially Patrick Winston, Rick Lathrop, and Gary Borchardt.

This work was supported, in part, by Digital Equipment Corporation, the National Science Foundation (Grant Nos. IRI-8805798 and IRI-8903034), and DARPA (Contract No. N00014-85-K-0124).

REFERENCES

Amarel, S. (1968). On the representation of Problems of Reasoning about Actions. in Webber, B. & N. Nilsson (Eds.) *Readings in Artificial Intelligence.* Tioga Publishing Company: Palo Alto, CA.

Birnbaum, L., M. Flowers & R. McGuire (1980). Towards an AI Model of Argumentation. *Proceedings of the First National Conference on Artificial Intelligence* Stanford, CA.

Bobrow, D. (1975). Dimensions of Representation. in Bobrow, D. & A. Collins (Eds.) *Representation and Understanding: Studies in Cognitive Science.* Academic Press: San Francisco, CA.

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

Brachman, R. & H. Levesque (1984). The Tractability of Subsumption in Frame-Based Description Languages. *Proceedings of AAAI-84* pp. 34-37. Austin, TX.

Brachman, R. & H. Levesque, Eds. (1985). *Readings in Knowledge Representation.* Morgan Kaufmann Publishers, Inc. : Los Altos, CA.

Card, S., J. D. Mackinlay & G. G. Robertson (1990). The Design Space of Input Devices. *Proceedings of CHI '90* 117-124. Seattle, WA.

Carroll, J. & M. B. Rosson (1990). Human Computer Interaction Scenarios as a Design Representation. *Proceedings of Proc. of HICSS-23: Hawaii International Conf. on System Sciences.* Los Alamitos, CA. IEEE Computer Society Press.

Carroll, J. & M. B. Rosson (1991). Deliberated Evolution: Stalking the View Matcher in Design Space. *Human-Computer Interaction* (this issue).

Carroll, J. M. & R. L. Campbell (1989). Artifacts as Psychological Theories: The Case of Human-Computer Interaction. *Behaviour and Information Technology* 8(4) 247-256.

Carroll, J. M. & W. A. Kellogg (1989). Artifact as Theory-Nexus: Hermeneutics Meets Theory-Based Design. *Proceedings of CHI '89* 7-14. Austin, TX.

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.

Conklin, J. & K. C. Yakemovic (1991). A Process-oriented Paradigm for Design Rationale. *Human-Computer Interaction* (this issue).

Fischer, G., A. Lemke, R. McCall & A. Morch (1991). Making Argumentation Serve Design. *Human-Computer Interaction* (this issue).

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

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

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

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

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

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

Lee, J. (1991a). *Decision Rationale Management System: Computational Representation and Use of the Reasons for Decisions.* Ph.D. Thesis forthcoming, Department of Electrical Engineering and Computer Science, MIT.

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

Lewis, C., J. Rieman & B. Bell (1991). Problem-Centered Design for Expressiveness and Facility in a Graphical Programming System. *Human-Computer Interaction* (this issue).

Lowe, D. (1986). SYNVIEW: The Design of a System for Cooperative Structuring of Information. *Proceedings of Computer Supported Cooperative Work* Austin, TX.

MacLean, A., R. Young, V. Bellotti & T. Moran (1991). Questions, Options, and Criteria: Elements of a Design Rationale for User Interfaces. *Human-Computer Interaction* (this issue).

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

Marshall, C. (1987). Exploring Representation Problems Using Hypertext. *Proceedings of Hypertext '87*

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.

McCall, R. (1990). PHIDIAS: A PHI-based Design Environment integrating CAD Graphics into Dynamic Hypertext. *Proceedings of European Conference on Hyupertext* Versaille, France.

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

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

Newman, S. & C. Marshall (1990). Pushing Toulmin Too Far: Learning From an Argument Representation Scheme. Xerox PARC, Technical Report.

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

Shum, S. (1991). Cognitive Dimensions of Design Rationale. *submitted to HCI'91*

Smolensky, P., B. Fox, R. King & C. Lewis (1988). Computer-Aided Reasoned Discourse or, How to Argue with a Computer. in Guindon, R. (Eds.) *Cognitive Science and Its Application to Human-Computer Interaction.* pp. 109-162. Ablex: Norwood, NJ .

Streitz, N., J. Hannemann & M. Thuming (1989). From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces. *Proceedings of Hypertext '89* pp. 343-364.

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

Winston, P. (1984). *Artificial Intelligence.* Prentice Hall: Reading, MA.

Yakemovic, K. C. B. & J. Conklin (1990). Observations on a Commercial Use of an Issue-Based Information System. *Proceedings of Computer Supported Cooperative Work* LA, CA.