# Design Rationale for Computer Supported Conflict Mitigation during the Design-Construction Process of Large-Scale Civil Engineering Systems

by

Feniosky Avelhermí Peña-Mora

Ingeniero Civil, 1987
Universidad Nacional Pedro Henriquez Ureña

Post-Grado en Administration de la Construction, 1988
Universidad Nacional Pedro Henriquez Ureña

Master of Science in Civil Engineering, 1991
Massachusetts Institute of Technology

**Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of**

**Doctor of Science in Civil Engineering Systems**

**at the**

**Massachusetts Institute of Technology**

**September 1994**

Author_________________________________________
Department of Civil and Environmental Engineering
August 12, 1994

Certified by_________________________________________
Robert D. Logcher
Professor of Civil and Environmental Engineering, Thesis Supervisor

Certified by_________________________________________
D. Sriram
Principal Research Scientist, Thesis Supervisor

Accepted by_________________________________________
Joseph M. Sussman
Chairman, Departmental Committee on Graduate Students

**Design Rationale for Computer Supported Conflict Mitigation during the Design-Construction Process of Large-Scale Civil Engineering Systems**

by

Feniosky Avelhermí Peña-Mora

Submitted to the Department of Civil and Environmental Engineering on August 12, 1994, in partial fulfillment of the requirements for the degree of Doctor of Science in Civil Engineering Systems

## Abstract

The development of large scale engineering systems requires the collaboration of numerous specialists. Their decisions reflect different perspectives of a project and these different perspectives typically lead to many conflicts. These conflicts, if not resolved early, create more expensive designs, delays in the design-construction process, and compromises in the final product. Thus, a fundamental issue in collaborative engineering is conflict mitigation. A set of case studies suggests that some of the conflicts during the process stem from the lack of information that certain specialists have about other specialists' objectives and reasons for rejecting or accepting a given alternative (i.e, design rationale). Yet, if the design rationale of all participants is made available to others, designers can become overwhelmed with data and its complexity. Thus, there is a pressing need for systems which help designers capture, interpret, and easily utilize this data when conflicts are detected. This thesis presents research on the representation, use, and communication of design rationale for conflict mitigation in a collaborative environment. This research is based on the view that: 1) the designers' perspectives are expressed in their design rationale; 2) a system for capturing the design rationale needs to represent and manage design intent evolution, artifact evolution, and relationships between intents and between intent and artifact; 3) a design rationale system needs to capture its information in a non-intrusive manner by providing part of the design rationale; and 4) a system for conflict mitigation needs to provide active computer support for the negotiation between multiple participants. Based on these requirements, this work develops and demonstrates DRIM (Design Recommendation and Intent Model) as an ontology for design rationale, and further develops SHARED-DRIMS (SHARED-Design Recommendation and Intent Management System) as a system for conflict mitigation in a collaborative environment. Testing of both the model and the system have been limited to small-scale problems dealing with conceptual design. The approach is potentially extensible to apply throughout the life-cycle of large scale design-construction problems.

Thesis Supervisor: Robert D. Logcher
Title: Professor of Civil and Environmental Engineering

Thesis Supervisor: D. Sriram
Title: Principal Research Scientist

# Acknowledgments

I wish to thank:

- Professor Robert D. Logcher for his support and guidance during these five years at MIT, and more importantly, for allowing me to be his apprentice.
- Professor Duvvuru Sriram for his useful comments and insight both in the classroom and in the DICE group, for his human touch and grace, as well as, for mentoring me all these years. "¿Como esta señor?".
- All the other members of the doctoral committee for their valuable feedback and support during this process: Prof. Connor, Dr. Germaine, Dr. Cherneff, Mr. McManus, Dr. Kumar, and Dr. Sycara.
- Some members of the Civil and Environmental Department for their feedback and encouragement: Prof. Lerman, Prof. Williams, and Dr. Shyam-Sunder.
- The Ford Foundation for its financial support and for the opportunity to meet good and young researchers that were in my situation.
- Gorti Sreenivasa Rao for all his advice, support, and intelligent discussions.
- Jonathan Cherneff for his continual support, encouragement, and wisdom.

- Anil Chakravarthy for his invaluable explanations, advice and time.
- Jintae Lee for introducing design rationale as a research area.
- Dr. Mikio Shoji, Atsuo Nishino, Seiji Morikawa and other members of Kajima Corporation for their encouragement and support.
- Joan McCusker, Elaine Healy, Patricia Vargas, Patricia Dixon, Cynthia Stewart, Jessie Carty, Stephanie George, Danielle Severino, and Muriel Frederick for their help in all the administrative and "real world" matters.
- All IESL members for their help and comments: Jesus Favela, Albert Wong, Ashok Gupta, Cheekian Ooi, Patrick Kinicutt, Nabha Rege, Kevin Amaratunga, and Allan Brik.
- Room 1-241's family for sharing their space and time: Andy, Leni, David, and Ann.
- All Club Latino members for their support outside of academics: Jaime, Monica, Maria Suzana, Santiago, Miguel, Janet, Joaquin, Octavio, Antulio, Douglas, Adriana, Isela, Alan, Victor, Pedro, Jose Carlos, Brenda, Geraldo, Ante, Marcos, and Luis Alberto.
- All my "Hi" MIT friends for changing my view that no one cares to say "Hi, How are you?" in this place.
- Next to last, but very important, Shimizu Corporation, the Intelligent Engineering Systems Laboratory (IESL), and the Civil and Environmental Engineering Department for their additional financial support.
- Most importantly, my family, my wife, and my future child for bearing with me all my sorrows and joys.

# Dedication

A los pilares de mi vida:

- **Mami** por sembrar en mi una vision positiva cada vez que tenia que conseguir algo muy importante. Siempre me decias: "Lo unico que puedes es ganar. Si tratas y no lo obtienes, te quedas igual –tu no lo tenias como quiera. Si tratas y lo obtienes, ganas. Por tanto solo puedes ganar."
- **Minin** por quererme tanto y apoyarme en todo lo que hago.
- **Mis Hijos** por que se que ustedes seran mi orgullo y mi mejor trabajo.

To the pillars of my life:

- **Mother** for planting in me a positive vision of life whenever I wanted to to get something important. You always said: "You can only win. If you try and don't get it, you stay the same – you don't have it anyway. If you try and get it, you win. You see, you can only win."
- **Minin** for loving me so much and providing me support in everything I do.
- **My Children** because you will be my pride and my best work.

# Contents

# List of Figures

# Notation

- $P_{horizontal}$ is the soil pressure on the horizontal direction.
- $P_{vertical}$ is the soil pressure on the vertical direction.
- $\phi$ is the angle of internal friction of the soil. Its tangent is equal to the coefficient of intergranular friction, which can be determined by appropriate laboratory tests.
- $c$ is the coefficient of earth pressure at rest.
- $\tan^2$ is the tangent of an angle to the power of two
- $\mathbf{A} = \{(\mathbf{t}_i, \mathbf{a}_i, \mathbf{v}_i)\}^*$ is a set of symbols. Each of these symbols are explained in the text of the thesis as they occur.
- (**Designer**, **oid**, A, M, R, C) is an class description with a class identifier, sets of attributes, methods, relationships, and constraints.
- $\forall$ is the for all symbol.
- $\in$ is the in a set symbol.
- $\exists$ is the exists symbol.
- $\subset$ is the sub set symbol.

## NOTATION

- : is the such that symbol.

- . is a reference symbol. It implies the part of an object. For example, in $r_{\ell_m}.[e|b]$, . implies that $e$ or $b$ are part of $r_{\ell_m}$.

- [] is a reference symbol which implies grouping. For example, in $r_{\ell_m}.[e|b]$, [] implies the set of $e$ or $b$ that are part of $r_{\ell_m}$.

- $\Rightarrow$ is the implies symbol.

- $\equiv$ is the equivalent symbol.

- $d_\ell(i_\ell, C)$ is a justification called $d_\ell$ which is linked to an $i_\ell$ intent in a $C$ context.

- $\mathcal{O}(i_{\ell+1}|plan_{\ell+1}|a_{\ell+1})$ is an operator called $\mathcal{O}$ which takes as argument $i_{\ell+1}$ or $plan_{\ell+1}$ or $a_{\ell+1}$.

- | is the or symbol.

- $t_0$ is the time symbol. In this case, the symbol represents time 0.

# Chapter 1

# Introduction

*Managing ... conflicts typically involves complex phenomena, multiple issues, and institutions having differing objectives and responsibilities. The synthesis, understanding and communication of relevant data and the identification and evaluation of possible solutions with the aid of analytical and computer models is often proposed, and occasionally accomplished, to support one or more participants in a conflict management process.*

Daniel P. Loucks, *[Loucks, 1989]*

## 1.1 Overview

The development of large scale engineering systems requires the collaboration of numerous specialists. These specialists may reach conflicting decisions as a result of their different perspectives on a given problem. These perspectives are revealed in their design rationale. However, each designer brings considerable knowledge and performs a great number of tasks to come up with a design. If all of this information were available to all the designers, each one would be overwhelmed with data. Thus, a computer system is needed that will help designers to record and use information effectively from the whole group to achieve effective conflict mitigation.

Section 1.2 introduces the problem for the Architect-Engineering-Construction (AEC) industry. Then, in Section 1.3, the motivations for undertaking this research are presented. The goals sought and the scope of the research are presented in Section 1.4. The approach followed in the research is then described in Section 1.5. Finally, Section 1.6 outlines the organization that will be followed in this dissertation presentation.

## 1.2 Research Problem

The development of large-scale engineering systems requires the collaboration of numerous specialists. During such a development process, up to as many as three hundred different specialty design firms, suppliers, and contractors may participate. This participation results in interactions among many different types of professionals. Their decisions reflect their different perspectives on a project, and these different perspectives typically lead to many conflicts. These conflicts, if not resolved early, create more expensive designs, delays in the schedule of the design-construction process, and compromises in the final product. Avoidance of these conflicts is typically accomplished with overly conservative designs, when applied to physical or functional objects which interact. In the next paragraphs, these various problems will be illustrated with two case studies.

The Boston Central Artery/Tunnel is a 7.8 billion dollar example of a large-scale engineering project [Sheridan, 1993]. It may eventually involve more than 150 main organizations during both the design and the construction phases. These organizations may work on the same or different parts of the project, and they must interact with each other. It is these interactions that exemplify the kinds of situations that can produce conflicts.

The design of the Central Artery segment that crosses the Charles River has generated many conflicts [Sheridan, 1993, Project, 1994]. There have been over 80 alternative layouts suggested for that crossing over a period of two and a half years. One of the reasons for such a large number of alternatives is that some of the participating organizations have recommended alternatives that have already been ruled out by other organizations. For example, the community organizations rejected the use of a bridge for the crossing, as suggested by the designers, and recommended the use of a tunnel. However, a tunnel had previously been rejected by the designers because it was not economical and involved large risks during construction. Then, after several iterations, the community organizations and the designers started exploring a combination of bridge and tunnel for the crossing. If the community organizations had had access to the information about the reasons for the designers' rejecting the tunnel, they could have started exploring other possibilities sooner instead of spending the time in a design alternative that was unsatisfactory from the design point of view both economic and ease of construction. Thus, some of the inefficiencies of the process stem from one group's lack of information about other groups' objectives and reasons for rejecting or accepting a given alternative, such information is termed as "design rationale".

This lack of information about organizations' design rationale can be attributed to deficiencies in their channels of communication. They use blueprints and specifications for intra and inter-organizational communication. Some of the more advanced organizations have begun to use CAD files for their internal communications. These communication mechanisms (blueprints, specifications, and CAD files) deal only with the characteristics of the artifacts. They do not transfer information about the objectives and the reasons for accepting or rejecting a design alternative. While such information is important to subsequent decision-making, it can only be surmised from these communications.

The failures to communicate are not limited to the designers of the Boston Central Artery/Tunnel. NYC's Bureau of Water Pollution Control has also recognized the problems due to the lack of communication about objectives. The Bureau has issued a handbook about conflict based on its experience during the upgrade of nine sewage treatment plants [City of New York Bureau of Water Pollution Control, 1980]. It developed a list of 2200 potential conflicts that should be avoided during the design of any of the plants. Most of these conflicts are based on the needs of different specialties to satisfy objectives that have not been explicitly stated. For example, where durability is the objective, the Bureau asks: 1) "Do the specifications provide for stainless steel angles where durability is required?"; 2) "Does the design provide for use of stainless steel railing to better withstand corrosion?"; and 3) "If data indicates fluctuating ground water levels, does design call for treated timber piles?" The specificity of these questions is impressive; and the common underlying theme is an explicit identification of issues that may help to avoid conflict. The handbook emphasizes the importance of learning from the experience of past design projects as it resulted on a conflict decrease on the development of sewage treatment plants. A study of the handbook suggests the value of a system that provides for capture, storage, and use of information from previous designs. These two cases – Boston Central Artery/Tunnel and the NYC's Bureau of Water Pollution Control – show the importance of communicating the design rationale (objectives and reasons) of the individual participants in the project and the rationale for its evolving overall plan.

These examples show that in today's design process, unfortunately, communication between the different participants has often lacked reference to the decisions and the reasoning that shaped the design, that is, reference to the design rationale. The AEC industry can benefit on two fronts from representing the design rationale explicitly: first, there will be savings in the life-cycle cost due to more effective communication, coordination and negotiation; and second, the quality of the final product will increase because the requirements or design intents are readily accessible for review.

## 1.3 Research Motivation

Nowadays, the computer plays an important role in almost every industry. In the AEC industry this role may be changing from one that emphasized computer graphics and numerical analysis to one of management of information [Gantz, 1989]. Traditionally, designers have used a wide range of knowledge and have performed a large number of tasks in selecting the characteristics of an artifact. If this information is made available to all designers working on a project, individual designers could become overwhelmed with data and its complexity. Thus, there is a pressing need for systems which help designers capture, interpret, and easily utilize this data.

As discussed in the previous section, design rationale can serve as an important basis for conflict mitigation. However, the information generated during the process of recording design rationale may be overwhelming. In keeping with the trend in information management systems, an intelligent support tool for use of this information can be perceived as an essential part of a conflict mitigation system. Here, the computer is used as an assistant that will monitor the design rationale as it is generated and will detect any potential conflict with other specialists' decisions.

This research is presented within the broader context of a set of support tools for representing and using design knowledge in a collaborative environment. It is a component of research on DICE (Distributed and Integrated environment for Computer-aided Engineering) [Sriram and Logcher, 1993]. The DICE framework is a set of computer-based tools for supporting design activities. These new tools are able to provide some of the design rationale stored in their knowledge base, to store the information in a persistent form and to allow for future modifications by other tools. The availability of such environments, the potential for capturing design rationale, and the pressing need for conflict mitigation have motivated this research.

## 1.4 Research Goals and Scope

The goals of this research are to represent, use, and communicate design rationale for conflict mitigation in a collaborative environment. To this end, the research is aimed at developing: an ontology for the design rationale; and a management system to capture, interpret, and easily utilize that ontology.

The model provides the constructs for defining all the elements that are involved in the design rationale. It not only helps to record the designed artifact itself, but also the proposals that introduced it, the objectives it is trying to achieve, and the justifications for selecting it.

The system for conflict mitigation addresses two critical research problems: first, design rationale has to be captured from humans or computers involved in the design of the project; and second, a system needs to reason about the captured rationale in order to provide support for conflict mitigation in a collaborative environment.

The scope of the system reasoning will be to automatically resolve known conflicts when solutions are available in its knowledge base, and to provide hypotheses about the reasons for unknown conflicts. These hypotheses, once verified by the designer, permit better coordination and negotiation during conflict resolution. This, in turn, will enhance communication during the design process and consequently increase the productivity of the AEC industry.

The scope of testing of both the system and the model is limited to small-scale problems dealing with conceptual design. The approach is potentially extensible to apply throughout the life-cycle of large scale design-construction problems.

## 1.5 Research Approach

In order to capture the design rationale for conflict mitigation, the design process and its evolution must be analyzed. With this perspective in mind, three case studies in design were performed (see Section 2.2). Based on these cases, a set of requirements was developed for a design rationale framework for conflict mitigation system. Then, a survey of current models or systems that capture design rationale was performed. A Design Recommendation-Intent Model (DRIM) was proposed as an ontology for representing the design rationale. SHARED-DRIMS was then developed as a system for conflict mitigation through the capture and management of the design rationale. After the model was developed, three more case studies were performed to validate it and to provide the test-bed for SHARED-DRIMS.

## 1.6 Thesis Organization

- **Chapter 2** presents the background information for the research described in the remainder of the thesis. The design process is analyzed from the conception of a need to the functioning of an artifact. It describes the six case studies performed during the research to create the system requirements and to validate the system. The set of requirements for a conflict mitigation system is presented.

- **Chapter 3** outlines the use of computer supported collaborative engineering in terms of the DICE framework, with a focus on the overall architecture and organizational view. It explains the methodology for information modeling combining object-oriented methodology and its use in the representation of semantic networks. Finally, the SHARED model is presented, providing the basis for the representational structure of the DRIM model.

- **Chapter 4** provides a survey of related work on design rationale with focus on the number of participants that the models support and the computer involvement on providing part of the rationale. In addition, a survey of related work on conflict mitigation is presented with a grouping of user-driven models and automated models. The user-driven models or systems take a set of options presented by the designers and points out the best one. The automated systems provide the options to the designers, but does not necessarily interact efficiently with them.

- **Chapter 5** describes a model which incorporates the different elements of design rationale. This chapter presents the DRIM primitive classes, the DRIM relationships, the process of design with the DRIM objects, and the development of design rationale traces.

- **Chapter 6** presents the functionality of the system that helps in the capture and utilization of design rationale for conflict mitigation. The different sections of this chapter cover the SHARED-DRIMS architecture including the base modules and the design rationale modules. Other sections present the conflict mitigation process with conflict detection, the generation of hypotheses for the causes of the conflicts and the hypotheses communication.

- **Chapter 7** illustrates the system at work through an example. The Sollecks River bridge example is described, focusing on the selection of the structural material and form. The conflict mitigation between the different participants in the process is provided.

- **Chapter 8** summarizes the main issues discussed in the thesis. It also discusses future research issues.

## 1.7 Conclusions

The AEC industry faces a problem in terms of mitigating conflict. This problem arises from the collaboration of many specialists with different perspectives, which conflict in the realization of the project. One hypothesis is that these conflicts could be mitigated by making the design rationale of the participants available to everyone participating in the process. However, this information can be overwhelming for the designers, and therefore the need for systems that help designers capture and use that information must be addressed.

# Chapter 2

# Requirements Analysis

*The first step in developing anything is to state the requirements. This applies just as much to leading-edge research as to simple programs and to personal programs, as well as to large team efforts.*

Rumbaugh, Blaha, Premerlani, Eddy and Lorensen,
*[Rumbaugh et al., 1991]*

## 2.1 Introduction

Engineering design and construction are activities that humans have performed since early times. However, these areas have not been perceived as systematic processes which could be subject to comprehensive analysis and improvement until the past few decades. The understanding of the design process is important for representing and capturing design rationale. Thus, the use of case studies about design will be relevant in presenting participant's interactions, information, and tasks. Section 2.2 describes six case studies performed during the research to create the system requirements and to validate the system. Section 2.3 presents a general description of the design process, from the conception of a need to the functioning of an artifact, based on the case studies and the design literature. Section 2.4 presents a set of requirements for a conflict mitigation system. Finally, Section 2.5 provides a summary of this chapter.

## 2.2 Case Studies

The understanding of the design process is important for representing and capturing design rationale. The use of case studies about design sheds light into the interactions among participants, the type of information used in design and how they are used. Lastly, it shows the different steps performed by the designers. The following sections present six case studies. These examples are as follows: the selection of an earth retention system, the design of a valve, the design of a flume, the structural design of a building, the structural design of a bridge, and the selection of a river crossing.

### 2.2.1 Earth Retention System Design

A developer and an architect of a building hire a construction consulting firm to assist in the selection of an earth retention system [Becker, 1989]. The building has two levels of parking extending 24 feet below the existing site level which is at an elevation of 57.0 feet. The design of the earth retention system needs to take into consideration the fact that the building is designed to be located in a site where an existing five-story brick building, which is very close to the new building, needs to remain at its current location. In addition, the designer of the retention system has to consider the potential effects of a river that passes close to the project site.

Based on these facts, the construction consulting firm has been asked to provide its recommendations for the earth retention system after completing the following process:

1. State the design requirements.
2. Present the alternatives that satisfy the requirements.
3. Present the reasons that explain how the alternatives satisfy the requirements.
4. Analyze the reason presented for each alternative.
5. Select the alternative that is the most appropriate.

With this process in mind the consultant starts by presenting six of the requirements sought with the earth retention system. These requirements are to: 1) minimize the cost of the earth retention system; 2) minimize project completion time; 3) increase reliability of the earth retention system; 4) control groundwater flow; 5) resist soil and water pressure; and 6) reduce the impact on the adjacent building. The developer and the architect provided the consultant with two alternatives which may satisfy the requirements: 1) Z-sheeting piles, and 2) structural slurry wall. This scenario shows that designers start with a set of requirements that they need to satisfy. It also shows that for certain requirements, such as minimize, increase, and reduce, comparisons between alternatives are needed. In this particular design, the alternatives taken were only two. Now, the consultant searches for justifications that establish a relation between the alternatives and the requirements. These justifications are:

**Alternative I:** Z-Sheeting

1. REQUIREMENT: Minimize the cost of the earth retention system
   - Z-Sheeting can be provided by many capable and established suppliers.
   - Z-Sheeting requires no waterproofing.
   - Z-Sheeting may require expensive underpinning of adjacent structures.

- Z-Sheeting needs careful installation for system to work as expected.

2. REQUIREMENT: Minimize project completion time
   - Z-Sheeting can be provided by many capable and established suppliers.
   - Z-Sheeting may require expensive underpinning of adjacent structures.
   - Z-Sheeting needs careful installation for system to work as expected.

3. REQUIREMENT: Increase reliability of the earth retention system
   - Z-Sheeting can be provided by many capable and established suppliers.
   - Z-Sheeting is commonly used in this area
   - Z-Sheeting may require a clear driving path

4. REQUIREMENT: Control groundwater flow
   - Z-Sheeting prevents water table lowering.
   - Z-Sheeting provides a water seal both temporary and permanent

5. REQUIREMENT: Resist soil and water pressure
   - Z-Sheeting resists the active earth pressure given by the formula

     $$P_{horizontal} = P_{vertical} \tan^2(45^o - \phi/2) - 2c\tan^2(45^o - \phi/2)$$

6. REQUIREMENT: Reduce the impact on adjacent building
   - Z-Sheeting may require expensive underpinning of adjacent structures.
   - Z-Sheeting can be provided by many capable and established suppliers.
   - Z-Sheeting produces settlement-causing vibrations during installation.
   - Z-Sheeting prevents water table lowering.

**Alternative II:** Structural Slurry wall

1. REQUIREMENT: Minimize the cost of the earth retention system

- Structural slurry wall can be provided by many capable, established suppliers
- Structural slurry wall requires no waterproofing.
- Structural slurry wall can be used as permanent foundation wall.
- Structural slurry wall has the highest cost of all the alternatives per linear foot.

2. REQUIREMENT: Minimize project completion time
    - Structural slurry wall takes a lot time to construct due to low production rate
3. REQUIREMENT: Increase reliability of the earth retention system
    - Obstruction on the work area does not cause major problem
    - Structural slurry wall can be provided by many capable, established suppliers
4. REQUIREMENT: Control groundwater flow
    - Structural slurry wall prevents water table lowering.
    - Structural slurry wall provides a water seal both temporary and permanent
5. REQUIREMENT: Resist soil and water pressure
    - Structural slurry wall resists the active earth pressure given by the formula $P_{horizontal} = P_{vertical}\tan^2(45^o - \phi/2) - 2c\tan^2(45^o - \phi/2)$
6. REQUIREMENT: Reduce the impact on adjacent building
    - Structural slurry wall prevents water table lowering.

This itemized account of alternatives, requirements, and justifications shows that designers use justifications as the linking elements between design artifacts and requirements.

The justifications are used as supporting arguments to the hypothesis that an alternative satisfies the requirements. In this design case, the designer analyzes each of the reasons related to the requirement. From this analysis, the professional determines that the Z-sheeting has a lower probability for meeting the requirement of reducing the impact on adjacent building while it has a greater chance of satisfying the requirement of minimizing the cost. Similarly, the structural slurry wall satisfies the requirement of reducing the impact of adjacent structures better, while it is not good at satisfying the reducing cost requirement.

After the designer performed the analysis s/he gives the highest priority to the goal of reducing impact on adjacent buildings. The reason given for this ranking is that the impact on adjacent buildings is a high legal risk that could cause the project to be stopped. In addition, the cost of repairing adjacent buildings was difficult to estimate until the problem occurs, since the documentation and the specification of existing buildings were not readily available. By designers providing a higher priority to a requirement, they are saying that not all the requirements are as important. Some of the requirements are needed while others are good to have.

Based on the priority of the goals and the analysis of the alternatives, the designer selects Alternative 2 (Structural Slurry Wall). One of the justifications that supports this decision is the assumption that this earth retaining system does not produce settlement-causing vibrations during installation. The second justification is that the structural slurry wall is as good as the Z-sheeting alternative in preventing the water table lowering thus achieving good groundwater control.

On the other hand, the structural slurry wall presents deficiencies in achieving the goal of cost minimization. This goal received low ranking in the designer priority list and did not have great influence in the decision. Thus, the rationale included: the requirements of the design, the assumptions made during the design process, the two alternative designs, the reasons of each alternative, and the selection process of choosing one design over the other based on the reasons and the ranking of the requirements.

This design case shows the dynamic structure of the decision process. A set of requirements that at the beginning had equal importance have now been ranked. This ranking in turn can affect the decision of selecting one alternative or another. Thus, it is necessary to capture the shift in importance of the elements involved in the decision. In the above design case, if construction cost were the only driving factor in the decision, the selection of the system would have been scheme 1 (Z Sheeting).

After a solution alternative has been selected, a record has to be maintained not only of the selected alternative but also of the process by which this alternative was selected together with all the other alternative solutions considered in the decision problem. In the above design case, schemes 1 and 2 have to be stored along with all the explanations given in the previous paragraphs. When a question about certain requirements arise, the decision process can be searched and explanations for the problem can be found.

For example, the owner of the project was given certain preliminary cost estimates of the project before the design of the complex started. Then, after two months, when the earth retaining system has been selected, the owner receives new cost estimates. These new estimates vary considerably from the previous one and s/he asks why the estimates are so different from the initial ones. In this case, the architect needs to give reasons for the variance. If the project team had captured all the information described previously, they need search only for the decisions associated with project cost.

### 2.2.2 Valve Design

An entrepreneur assigned a design team to design an artifact that would redirect and stop the flux through a network of tubes within a larger device [Kumar, 1992]. This artifact would be used in laboratories where different liquids and gases circulate through a network of tubes. This case study presents the different steps that are performed from the study of market needs to manufacturing process. It also highlights some of the main issues raised during the design process, i.e. justification of cost and lack of achieving a requirement.

This case study was performed and documented for this dissertation research. However, it cannot be included in this dissertation because the designer was seeking a patent and the details of the design could not been given out. Only the results form the case study will be presented on the following paragraphs.

#### Summary

The scenario that was studied demonstrated the dynamic structure of the design-decision process. The following is a summary of what transpired in this scenario:

- the requirements for the design were elaborated at different stages of the design. Exploration and discovery comes together since some of the ideas developed as the design proceeded. The designers never started with the idea of having different markets to which the design could be sold such as biological industry and chemical industry. That idea evolved as the design developed.
- designers had limited knowledge and needed the interaction and communication with other professionals. For each of the material types, the designer called on other sources for information such as a ceramic manufacturer, with whom he queried the manufacturability of the design.

- designers used prototypes to clarify concepts. The physical artifact was able to demonstrate how the concepts will work under real world conditions.

- various design alternatives were arrived at that could satisfy the requirements to certain degrees.

- within the realm of the requirements the rationale of each design were explored.

- each of the requirements for the design was given a priority level. This ranking in turn affected the decision of selecting one alternative design over the other.

- designers used concepts from other fields based on similarity of properties. In this case, the designers tested the use of ionic coating because of its success in pumps.

### 2.2.3 Flume Design

This case refers to the design of a flume to be used for testing the effect of soil erosion in the contamination of river basins [Andrews, 1993]. The design has to allow for the settlement and suspension of the soil particles. This is necessary to determine the type of flow that causes erosion of naturally sedimented soils. In this design, the principal professionals involved are three geotechnical engineers, one of whom was in charge of constructing the flume (the contractor) and another was in charge of testing the erosion theory (the user). All the geotechnical engineers are responsible for designing the flume so that the relationship between the effective stress-strength parameters of sediments and the critical shear stress required to initiate erosion can be established. The constructor is responsible for constructing the flume so that it satisfies the geotechnical engineers' specifications. The user is responsible for performing the tests required and demonstrate the theory. As this case shows, the constructor's work depends on the geotechnical engineers' and the geotechnical engineers' work depends on the tests that the user wants to perform. This interaction requires considerable coordination and communication, in order to achieve the desired requirements. Below, the reasoning process of the geotechnical engineers is recorded.

The geotechnical engineers need to establish the relationship between the effective stress-strength parameters of sediments and the critical shear stress required to initiate erosion. To achieve that *goal*, they present a proposal that contains –as a recommendation– a plan with the following *goals*: 1) use of marine clay, specifically Boston Blue clay; 2) perform erosion tests on a deposited bed; and 3) perform strengths tests on the same bed. The justifications of using such a plan are: 1) it proposes the use of a cohesive sediment that is important for the contamination that may result if it is eroded; 2) it allows the investigation of erosion on normal and uniform bed sediments; and 3) it allows the measurement of the strength of the soil under different stresses. Figure 2-1 shows the representation of this first proposal. Figures 2-2 shows the proposal hierarchy as developed by the geotechnical engineers.

The proposals introduced the use of a linear flume at the beginning. As the design progressed, the flume was changed to a linear flume with re-circulating pump and finally to a rotating annular flume. All these proposals were introduced as consequence of new intents evolving as decisions were made. One observation made by the designers was that these new intents and proposals had some relationships with the old information. The concept of linear flume was developed since water was needed to run constantly through the flume. However, there were practical problems with it. The water condition at the ends would have been different from the water condition on the flume. This problem rule out the use of linear flume. Thus, a new concept was sought. This concept was the use of a rotating annular flume. This new concept was then evaluated with respect to all the intents that the linear flume needed to satisfy. Here, it can be seen that the design rationale was used for defining the context of new concepts. Design rationale would be also useful for defining why a rotating annular flume was selected. Without that information, other participants on the project may arrive at wrong conclusions about the need of such a flume.

It was also found in this case study, that participants requested design rationale information when they were negotiating changes. For example, one of the geotechnical engineer proposed a re-circulating pump to make the water run through the flume. However, one of the other engineers opposed this decision. They entered into a negotiation process about the use of re-circulating pump. During that process, they presented reasons against and in favor of the decision. One of the reason in favor was that re-circulating pump would allow the modeling of the actual length of the river. This reason implies that modeling the actual length of the river was an intent that needed to be satisfied. One of the reason against the use of re-circulating pump was that it would destroy the aggregate particles. This reason in turn implies that accurately modeling the particles on the river was another important intent. In both cases, the geotechnical engineers presented intents that need to be satisfied in order to achieve a satisfactory design. The way they resolved the conflict was by selecting a solution that satisfied both designers. However, this resolution was only achieved when the intents were explicitly stated for everyone to use. Thus, the explicit presentation of intents would allow a quicker resolution of the conflicts.

### 2.2.4 Building Design

A structural engineer has to design a one story building with a basement and structural capacity for supporting 2nd and 3rd floor future additions. The structural engineer interacts with an architect, a contractor, the owner, and a mechanical engineer during the structural

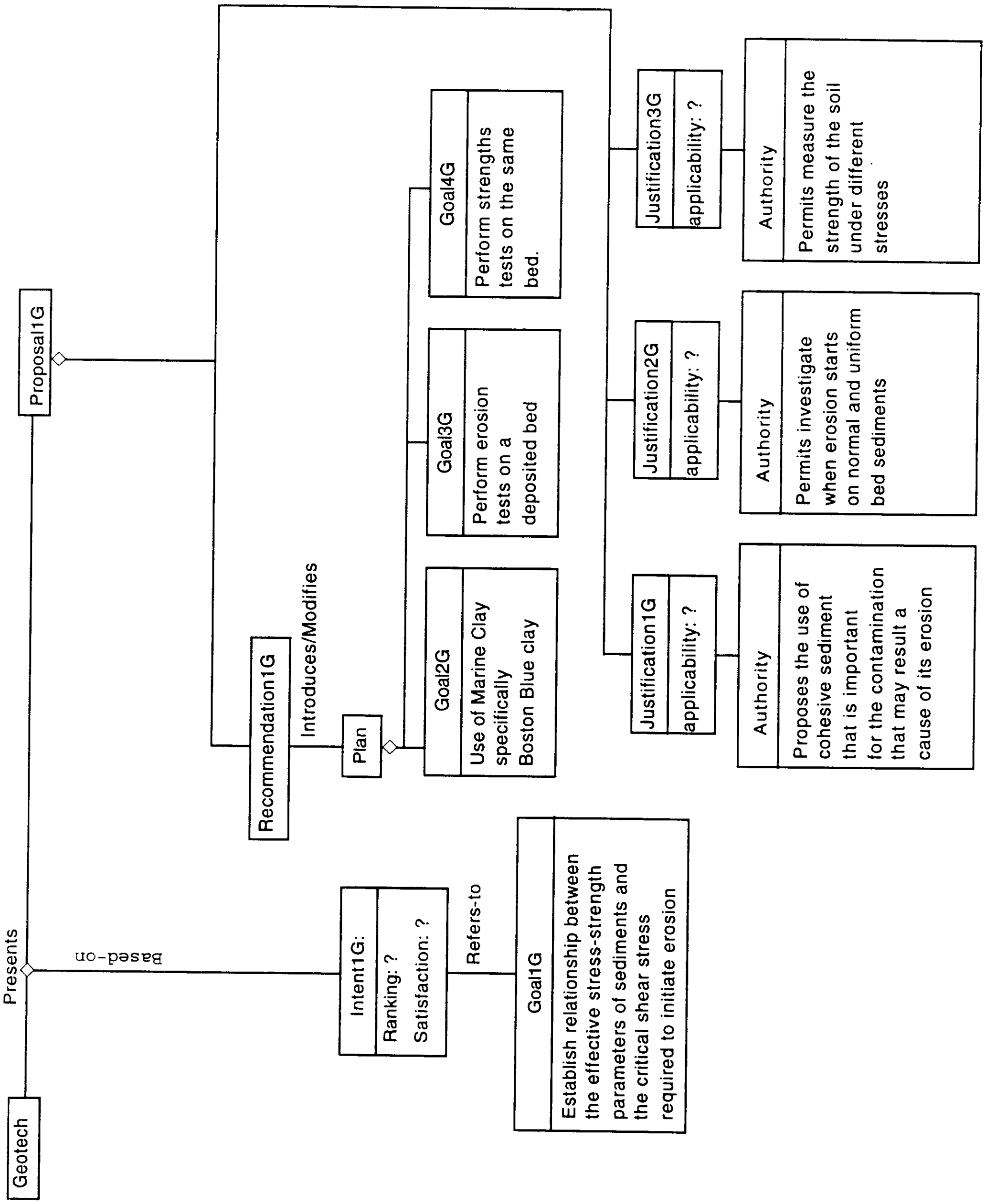
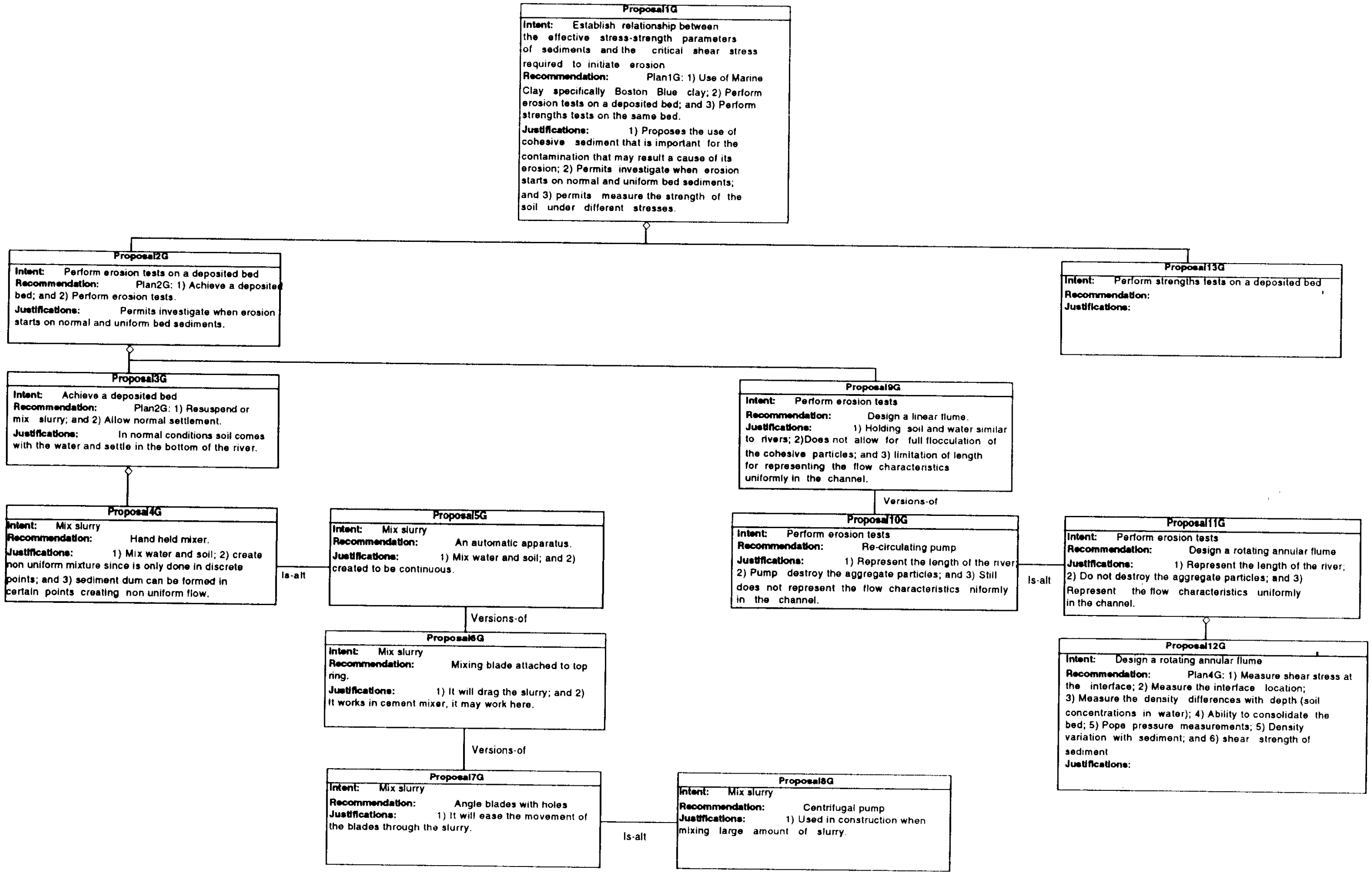

Geotech
Presents
Proposal1G
Based-on
Intent1G:
Ranking: ?
Satisfaction: ?
Refers-to
Goal1G
Establish relationship between the effective stress-strength parameters of sediments and the critical shear stress required to initiate erosion
Recommendation1G
Introduces/Modifies
Plan
Goal2G
Use of Marine Clay specifically Boston Blue clay
Goal3G
Perform erosion tests on a deposited bed
Goal4G
Perform strengths tests on the same bed.
Justification1G
applicability: ?
Authority
Proposes the use of cohesive sediment that is important for the contamination that may result a cause of its erosion
Justification2G
applicability: ?
Authority
Permits investigate when erosion starts on normal and uniform bed sediments
Justification3G
applicability: ?
Authority
Permits measure the strength of the soil under different stresses

Figure 2-1: First Proposal of Geotechnical Engineers



**Proposal1G**
**Intent:** Establish relationship between the effective stress-strength parameters of sediments and the critical shear stress required to initiate erosion
**Recommendation:** Plan1G: 1) Use of Marine Clay specifically Boston Blue clay; 2) Perform erosion tests on a deposited bed; and 3) Perform strengths tests on the same bed.
**Justifications:** 1) Proposes the use of cohesive sediment that is important for the contamination that may result a cause of its erosion; 2) Permits investigate when erosion starts on normal and uniform bed sediments; and 3) permits measure the strength of the soil under different stresses.

**Proposal2G**
**Intent:** Perform erosion tests on a deposited bed
**Recommendation:** Plan2G: 1) Achieve a deposited bed; and 2) Perform erosion tests.
**Justifications:** Permits investigate when erosion starts on normal and uniform bed sediments.

**Proposal13G**
**Intent:** Perform strengths tests on a deposited bed
**Recommendation:**
**Justifications:**

**Proposal3G**
**Intent:** Achieve a deposited bed
**Recommendation:** Plan2G: 1) Resuspend or mix slurry; and 2) Allow normal settlement.
**Justifications:** In normal conditions soil comes with the water and settle in the bottom of the river.

**Proposal9G**
**Intent:** Perform erosion tests
**Recommendation:** Design a linear flume.
**Justifications:** 1) Holding soil and water similar to rivers; 2)Does not allow for full flocculation of the cohesive particles; and 3) limitation of length for representing the flow characteristics uniformly in the channel.

**Proposal4G**
**Intent:** Mix slurry
**Recommendation:** Hand held mixer.
**Justifications:** 1) Mix water and soil; 2) create non uniform mixture since is only done in discrete points; and 3) sediment dum can be formed in certain points creating non uniform flow.

**Proposal5G**
**Intent:** Mix slurry
**Recommendation:** An automatic apparatus.
**Justifications:** 1) Mix water and soil; and 2) created to be continuous.

**Proposal10G**
**Intent:** Perform erosion tests
**Recommendation:** Re-circulating pump
**Justifications:** 1) Represent the length of the river; 2) Pump destroy the aggregate particles; and 3) Still does not represent the flow characteristics nlformly in the channel.

**Proposal11G**
**Intent:** Perform erosion tests
**Recommendation:** Design a rotating annular flume
**Justifications:** 1) Represent the length of the river; 2) Do not destroy the aggregate particles; and 3) Represent the flow characteristics uniformly in the channel.

**Proposal6G**
**Intent:** Mix slurry
**Recommendation:** Mixing blade attached to top ring.
**Justifications:** 1) It will drag the slurry; and 2) It works in cement mixer, it may work here.

**Proposal12G**
**Intent:** Design a rotating annular flume
**Recommendation:** Plan4G: 1) Measure shear stress at the interface; 2) Measure the interface location; 3) Measure the density differences with depth (soil concentrations in water); 4) Ability to consolidate the bed; 5) Pope pressure measurements; 5) Density variation with sediment; and 6) shear strength of sediment
**Justifications:**

**Proposal7G**
**Intent:** Mix slurry
**Recommendation:** Angle blades with holes
**Justifications:** 1) It will ease the movement of the blades through the slurry.

**Proposal8G**
**Intent:** Mix slurry
**Recommendation:** Centrifugal pump
**Justifications:** 1) Used in construction when mixing large amount of slurry.

Figure 2-2: Proposal Hierarchy of Geotechnical Engineers

design of the building. These interactions lead to many changes which in turn lead to conflicts among the participants. In this write up, the interactions are documented and the reasons for such interactions are explained.

1. The present roof is kept level to provide a floor for the future additions to the building. However, this design recommendation was not achieved until a round of negotiations between the architect, the owner and the structural engineer. Usually the steel used for the roof is sloped to get better drainage and the architect could have used tapered insulation to achieve this. The use of a flat roof was a compromise between good drainage and ease of construction for the next construction phase.

2. An existing tunnel posed a number of problems for architectural layout, structural scheme and mechanical ductwork space. The structural framing scheme was revised a number of times to work around the tunnel. First, the owner and architect thought that they could keep the tunnel without the building having access to it. Based on this, the architect changed the floor plan of the building so that there was no interference with the tunnel below. The owner then required that the building have access to the tunnel. The architect changed the layout to adapt to this new requirement. In the interaction between the architect and the owner, the need to share requirements across disciplines is shown. The architect includes the need to have tunnel access into his/her requirements. With that new requirement, the architect could produce a satisfactory solution to both his/her and the owner's requirements. In conclusion, conflicts can be mitigated if requirements are explicit and sharable.

3. The study made by the structural engineer revealed problems with the structural frame and the location of the tunnel. In addition, the contractor estimated that the access to the tunnel would increase the cost of the building significantly. Then, the

owner decided to relax this requirement. This decision also allowed for a better floor plan from the architect point of view.

4. Framing scheme was coordinated with the architectural designers to obtain somewhat even bay spacing in some areas and a big open space in one area. The reason for even bays was constructibility while the open space was based on aesthetics. The use of a big open space posed some problems. Location of a column was revised several times to fit the architectural layout. Another column was eliminated resulting in the need for a W30 beam, which in turn caused low headroom for mechanical ductwork. The mechanical engineers had to work around this beam and had to change their design accordingly.

5. The architect wanted to eliminate a ledge in the basement which was standing from an earlier construction. However, due to the cost of underpinning an existing stairwell footing, a compromised design was achieved in which the ledge stayed and the architect incorporated it into the architectural layout of the basement area.

6. The construction manager was surprised to know that steel weight came in at 10 lbs/sf instead of 6.5 lbs/sf that he had budgeted at the beginning of the project. He had not anticipated that designing for the future additions would involve such a great increase in steel members' weight. This change produced an increase on the cost of the project.

The process followed is similar to other projects. The project begins with schematic drawings. The structural engineer has to prepare a framing scheme (beam, column, etc.) with approximate sizes based on experience from other jobs. This activity lasted around two days with a lot of conversations over the phone. Contact was not possible at times and the structural engineer had to wait for the architect to get back to him. Then after schematic design had been performed, the architect would get back to the owner and the mechanical engineer. Some of the problems that arose were solved by phone and fax machines. Once the schematic was checked, the structural engineer went on to the detailed design. At this time, the steel is pre-bid based on rough quantity and price/unit of quantities because scope is defined and fast delivery is very important. This posed a serious problem because of the stage of the design. In that case the structural engineer was conservative and ordered steel that was not very likely to change during the early stage of the process. That same philosophy was used for detailed design. Designers performed detail design on the parts of the design that are not very likely to be changed. The areas that are likely to be changed always go back and forth several times with some down time for the need to contact the people that are involved in the decisions.

### 2.2.5 Bridge Design

This case refers to a bridge over the Sollecks River located in the Olympic peninsula in northwest Washington [White et al., 1972]. In this case, the focus was on the process of selecting the structural form and the material of the bridge with the interactions of the structural engineer, the environmental engineer, the geotechnical engineer and the contractor. The case has been augmented with information obtained from Dr. Ashok Gupta [Gupta, 1993] when the information could not be found in White et al., 1972.

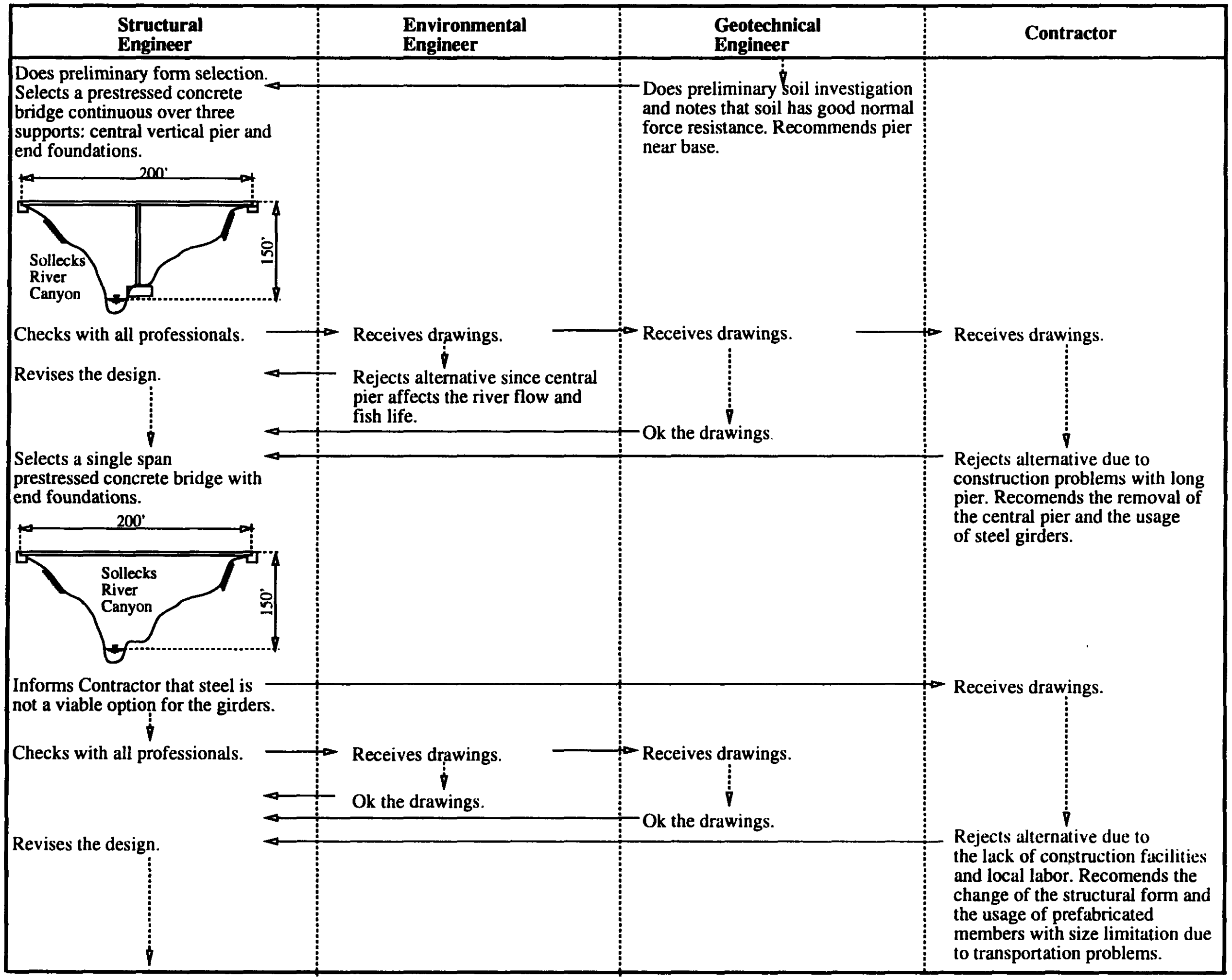
This case starts with the geotechnical engineer performing some preliminary soil testing (see Figure 2-3). During his investigation he determines that the soil has good normal force resistance and recommends the usage of piers near the base or ends. Then the structural engineer takes that information to select the form and material of the bridge. She decides to use prestressed concrete and a two span bridge with a vertical support and end foundations.

Subsequently, the structural engineer checks with all the other professionals involved in the project, namely the environmental engineer, the geotechnical engineer and the contractor (see first loop in Figure 2-3).

After receiving and reviewing the drawings, the environmental engineer rejects the design due to problems with the central pier. She argues that the fish life and river flow will be affected during construction. On one hand, the geotechnical engineer accepts the design. On the other hand the contractor rejects the design due to problems with the construction of the central pier and the usage of prestressed concrete. He recommends the removal of the central pier and the usage of steel girders instead of prestressed concrete. The structural engineer then uses the feedback obtained from the other participants to revise her design.

She then selects a prestressed concrete single span bridge. Again she checks with all the participants (see second loop in Figure 2-3). She informs the contractor that steel is not a viable solution since it violates objectives in her domain such as low maintenance cost for the bridge. Had the contractor had access to this information, this interaction could have been avoided. At this time both the environmental and geotechnical engineers accept the design. However, the contractor still rejects the design due to the lack of construction facilities for building such a long single span bridge using prestressed concrete. He recommends the use of prefabricated prestressed concrete and suggests that the structural engineer change the structural form of the bridge. This interaction could also have been avoided if the structural engineer had access to the contractor's reasons for rejecting the long pier the first time.

With this information, the structural engineer selects prefabricated prestressed concrete as the material and a three span bridge with two vertical piers as the structural form (see



Structural Engineer: Does preliminary form selection.

Geotechnical Engineer: Does preliminary soil investigation and notes that soil has good normal force resistance. Recommends pier near base.

Structural Engineer: Selects a prestressed concrete bridge continuous over three supports: central vertical pier and end foundations.

Structural Engineer: Checks with all professionals.

Environmental Engineer: Receives drawings.

Geotechnical Engineer: Receives drawings.

Contractor: Receives drawings.

Environmental Engineer: Rejects alternative since central pier affects the river flow and fish life.

Structural Engineer: Revises the design.

Geotechnical Engineer: Ok the drawings.

Contractor: Rejects alternative due to construction problems with long pier. Recomends the removal of the central pier and the usage of steel girders.

Structural Engineer: Selects a single span prestressed concrete bridge with end foundations.

Structural Engineer: Informs Contractor that steel is not a viable option for the girders.

Contractor: Receives drawings.

Structural Engineer: Checks with all professionals.

Environmental Engineer: Receives drawings.

Geotechnical Engineer: Receives drawings.

Environmental Engineer: Ok the drawings.

Geotechnical Engineer: Ok the drawings.

Contractor: Rejects alternative due to the lack of construction facilities and local labor. Recomends the change of the structural form and the usage of prefabricated members with size limitation due to transportation problems.

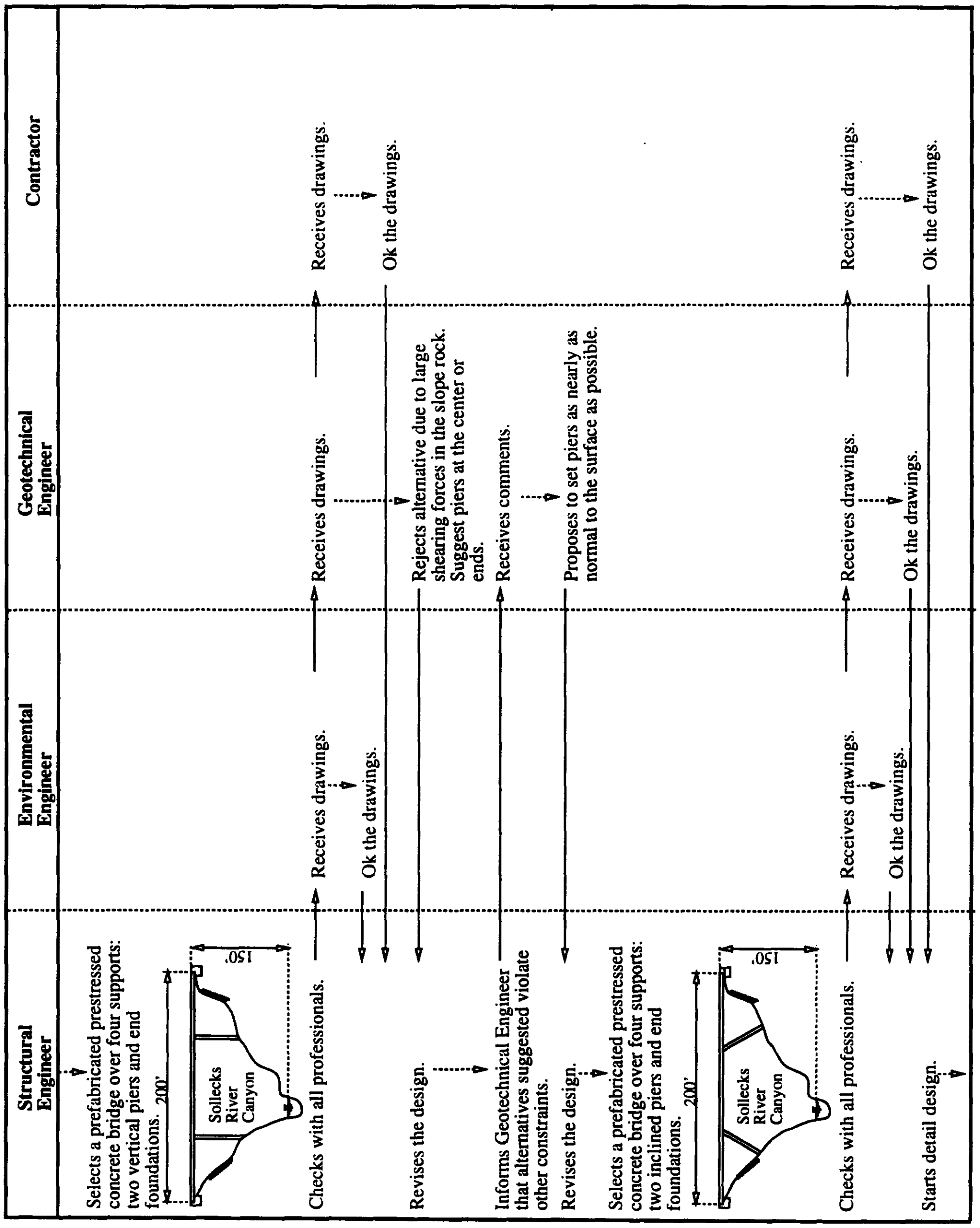
Structural Engineer: Revises the design.

Figure 2-3: Selection of structural form and material for Sollecks Bridge – Part I.

Figure 2-4). She checks one more time with all the participants in the project (see first loop in Figure 2-4). At this time both the environmental engineer and the contractor accept the design. The geotechnical engineer rejects the design due to the shearing forces that may develop in the sloped rock. He recommends the usage of piers at the center or at the ends. The structural engineer revises the design and informs the geotechnical engineer that piers cannot be set on the center or ends. The geotechnical engineer then suggests that the piers be located perpendicular to the slope.

The structural engineer receives that recommendation and selects a three span bridge with two inclined supports. She again checks with all the professionals (see second loop in Figure 2-4). All the professionals accept the design and the structural engineer proceeds with the detailed design.

As this case demonstrates, there are some deficiencies with the process. First, the process takes a long time to deliver. Second, the process may produce poor quality products. In terms of delivery time, there are three main deficiencies. First, some information may be lost, such as information regarding the reasons why a specific recommendation was made. Second, the participants will have to regenerate the information lost if they are asked for it. Third, the process has interactions that could be avoided if information about why a design has been rejected or why a recommendation is made is available. For example, the selection of a single span bridge would not have been presented as an alternative, had the structural engineer known the reasons for the rejection of the long pier by the contractor. In terms of the quality, there is one main problem. Constraints set by participants through assumptions may be violated. Violations may go undetected during the initial stages and may be detected at a later stage. The changes at later stages will be made under strong time and cost constraints. Designers will tend to focus on the problem area and disregard the



| Structural Engineer | Environmental Engineer | Geotechnical Engineer | Contractor |
|---|---|---|---|
| Selects a prefabricated prestressed concrete bridge over four supports: two vertical piers and end foundations. | | | |
| Checks with all professionals. | Receives drawings. | Receives drawings. | Receives drawings. |
| | Ok the drawings. | | Ok the drawings. |
| Revises the design. | | Rejects alternative due to large shearing forces in the slope rock. Suggest piers at the center or ends. | |
| Informs Geotechnical Engineer that alternatives suggested violate other constraints. | | Receives comments. | |
| Revises the design. | | Proposes to set piers as nearly as normal to the surface as possible. | |
| Selects a prefabricated prestressed concrete bridge over four supports: two inclined piers and end foundations. | | | |
| Checks with all professionals. | Receives drawings. | Receives drawings. | Receives drawings. |
| | Ok the drawings. | Ok the drawings. | Ok the drawings. |
| Starts detail design. | | | |

Figure 2-4: Selection of structural form and material for Sollecks Bridge – Part II.

overall optimization of the product.

### 2.2.6 River Crossing Selection

The Boston Central Artery/Tunnel is a 7.8 billion dollar example of a large scale engineering project [Sheridan, 1993]. It involves more than 150 primary organizations during both the design and construction phases. These organizations may work on the same or different parts of the project, but they must interact with each other. These interactions bring about conflicts that must be resolved.

The design of the Boston Central Artery/Tunnel segment that crosses the Charles River has generated a lot of conflicts [Sheridan, 1993, Project, 1994]. There have been over 80 alternative layouts for that crossing over a period of two and a half years. One of the reasons for such a large number of alternatives is that some of the participating organizations recommended alternatives that had been ruled out by other organizations. For example, the community organizations rejected the use of a bridge for the crossing as suggested by the designers and recommended the use of a tunnel. However, a tunnel had previously been rejected by the designers because it was uneconomical and involved large risks during construction. Then, after several iterations, the community organizations and the designers started exploring a combination of bridge and tunnel for the crossing. Some of that time could have been saved if the community organizations had access to the information about the reasons for the designers rejecting the tunnel. They could have started exploring other possibilities instead of spending the time in a design alternative that was unsatisfactory for the designers. Thus, some of the inefficiencies of the process stem from the lack of information that certain organizations have about other organizations' objectives and reasons for rejecting or accepting a given alternative (i.e, design rationale).

This lack of information about organizations' design rationale can be attributed to deficiencies in their communication channels. They use blueprints and specifications for inter-organizational communication while some of the more advanced organizations use CAD files for their internal communications. These means of communication lack the transfer of information about the objectives and reasons for accepting or rejecting a design alternative.

### 2.2.7 Requirement Synthesis from Case Studies

The case studies of design have shed light on the process: the interactions among participants, the type of information used in design, how this information is used, and how iterative the process is. The following is a summary of the issues that emerged from the case studies and the characterization of the process. Note that, this study concentrated on the interaction and information transfer between various participants. For case study introducing individual design process see Sriram et al., 1992.

- *the design intents or requirements were elaborated at different stages of the design.* The design starts with an inadequate understanding of the actual design intents. This inadequacy is a consequence of the inherently incomplete, possibly inconsistent, and perhaps unrealizable nature of the human needs. However, the design intents are refined with better understanding of the needs, as the design evolves and more data about these needs are gathered and evaluated.

- *designers had specialized knowledge and other designers need interaction and communication to understand their designs.*

- *designers used prototypes to clarify concepts.* The prototype was used to demonstrate how the concepts work in the real world.

- *designers used concepts from other fields, based on similarity of properties or features.*

- *several design alternatives that could satisfy the design intents to various degrees were usually generated.*

- *justifications of why a given design alternative satisfied a design intent were presented in case the design alternative is questioned by other designers.* This procedure implies that during the evaluation of a design alternative the designer has a set of justifications that support the use of such design alternatives. However, these justifications are not presented unless there is a doubt about the adequacy of the use of such design alternatives.

- *design intents were rated, depending on the importance that they have for the designers.*

- *this ranking, in turn, affected the decision of selecting one alternative design over the other.*

## 2.3 Design/Construction Process

The overall design process is composed of various stages according to [Woodson, 1966, Ostrofsky, 1977, Serrano, 1987, Pahl and Beitz, 1988] and the case studies presented in Section 2.2. There can be variations between theories, but in general the process consists of the following steps shown in Figure 2-5 and described in the next paragraph.

The design process consists of a *feasibility analysis* stage in which needs are identified, evaluated, and justified, a *problem identification and formulation* phase which defines the

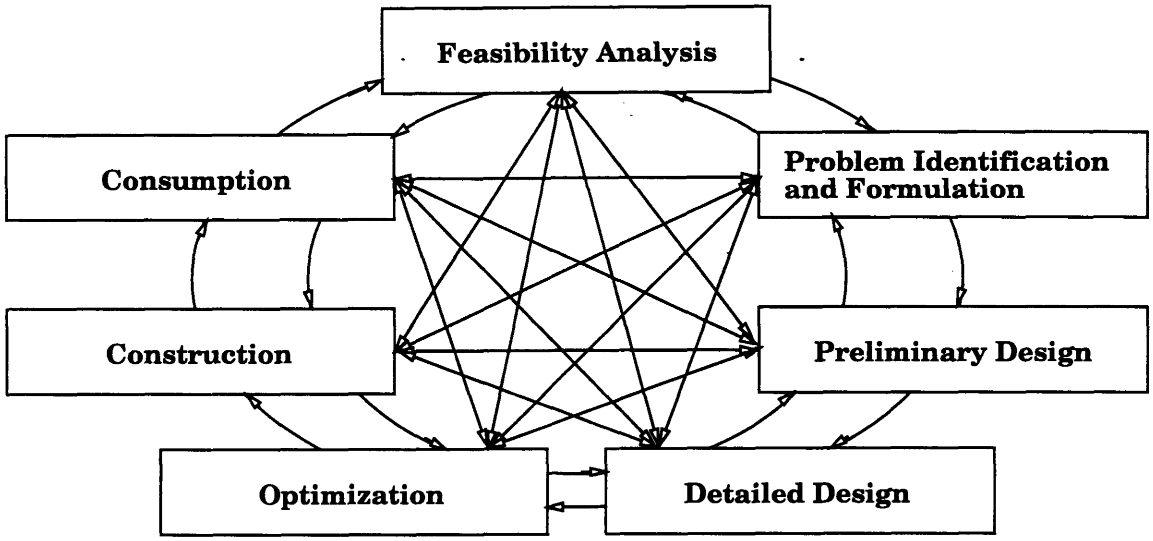


Figure 2-5: The Design Process.

scope and specifications of the problem, a *preliminary design* stage in which various design alternatives for the solution of the problem are generated and evaluated, a *optimization* phase in which design alternatives are refined through optimization of design parameters by sophisticated analysis and evaluation, a *detailed design* stage which lays out for construction the specifications about the arrangement, form, dimensions, and surface characteristics of all the system parts, a *construction* phase in which resources are converted into an artifact according to design specifications, and a *consumption* phase in which the artifact serves the needs for which it was designed.

This overall process can be defined as a collaborative-iterative decision-making activity organized to conceive the idea for, to prepare the description of, and to produce the plans by which resources are converted into artifacts or devices to meet human needs [Woodson, 1966], [Serrano, 1987],[Pahl and Beitz, 1988],[Sriram et al., 1989]. In the example described in Section 2.2.2, the human need was to make a better valve. In this regard, the designer conceived the idea of modifying a rotor valve, and preparing the description of the new rotor valve in terms of the material of the body and the rotor. Finally, the designer presented the plans in which the ionic coating and steel were put together to create the new valve.

The iterative characteristic of the design process is expressed by the inherently incomplete, possibly inconsistent and unrealizable nature of the human needs. These characteristics prevent designers from obtaining knowledge of the ideal solution and also resulting in an inadequate understanding of the actual design issues. Following the valve case, the need for producing a better valve is an incomplete description since there is no specification on what to make better for producing the valve.

Consequently, designers start by choosing a concept that abstracts what the consumer needs and have some way of satisfying those needs. In the case of the valve design, the designer choose the durability issue of the valve as the problem to be addressed. Using a *top-down* approach, this problem may be divided into sub-problems. These problems, in turn, may be further divided into smaller sub-problems. Durability could be divided into two subproblems: 1) reducing friction between the rotor and the body; and 2) increasing the hardness of the body so that scratches do not occur on the body. However, designers may also use a *bottom-up* approach in which detailed concepts are combined to make more abstract concepts. In order to design the valve for high temperature, the designer thought of combining ceramic and plastic to create a composite rotor so that the end thermal expansion is similar to the valve body thermal expansion.

Designer's initial assumptions are refined and a better understanding of the design is achieved as the design evolves and more data about the ideal solution and design issues are gathered and evaluated. For example, when exploring the ceramic valve, the designer introduced a new functionality of the body that was not important before. The body needed to be able to hold the entry tube to the ports. In stainless steel, the function of holding the tubes in place was not really important since it could be readily achieved by the body valve. In the ceramic valve design, the main problem is the threads, which are very difficult to make in ceramic.

In other words, designers approach design as an opportunistic activity [Banares-Alcantara, 1991]. They generate concepts using a *top-down* or *bottom-up* approach inter-changeably according to the information available at a given stage of the design and the applicability of either of those two approaches to that information.

### 2.3.1 Design as a Justificative Process

During the design process, designers need to justify their decisions. They use support knowledge to ensure that their selection is accurate to the best of their knowledge as well as a designer notebook to record their computations. This knowledge refers to the use of rules, cases, first principles, trade-offs, pareto optimal surfaces, constraint networks, catalog entries, authority commands. These types of justifications imply that their decisions fulfill the objective that they are searching to satisfy. These justifications happen in two ways. First, it goes from the objective to the justifications and then to the artifact. Second, it may go from the objective to the artifact and then the justifications are explored.

Either way, a set of justifications of how the sub-problems or concepts interact and how they may be solved or combined is inherent in the usage of the *top-down* and *bottom-up* approaches. Designers perform a set of steps that in turn will result in intermediate and terminal artifacts based on a set of justifications. These artifacts and the steps are usually the medium by which the designers communicate their ideas. However, designers need to express explicitly the justifications used during the design process in order to capture the design rationale.

### 2.3.2 Design/Construction as a Conflict Mitigation Problem

Myers [Myers, 1992] explains that the AEC industry is in a "lose-lose" situation when it comes to disputes resolution. This situation arises because of the lack of a mechanism for ensuring prompt detection and mitigation of conflicts between various participants. The erroneous decisions made in the early stages are less costly to fix if they are detected during the early phases. However, if the erroneous decision is left undetected or unresolved, the price for fixing it increases as time passes by.

It is only recently that researchers have tackled the integration of design and construction [Sriram et al., 1989] and [Howard et al., 1989]. Their goal is to provide two way continuity of information flow, without losses, between these processes and all the parties involved in the project. These efforts in cooperation support is backed a study done by [Will, 1991], which notes that the time spent in communication and documentation during design is 65% of the total work time of the participants. Favela [Favela, 1993] refers to a study conducted at HP's Fort Collins engineering workstation site [Will, 1991] in which traditional engineering tasks such as planning design and testing account for 35% of the time spent by engineers in a given project. The remaining time is spent in making sure that all the participants in the project understand what they have done. They need to communicate with other professionals 40% of their time and need to document their work 25% of their time. From our case studies, this communication and documentation refer to the designers' reasoning process –why they selected a given alternative or performed a given task– that needs to be communicated to others. They need to explain their intents (objectives, constraints, functions and goals) as well as the reasons for selecting a particular solution (artifact, a plan or another intent).

Myers [Myers, 1992] presents some methods for helping mitigate conflicts. These methods emphasize good preparation for the development process during the planning and design stages. They emphasize competent specialists, adequate budgeting, risk assessment, and value engineering. Then, during construction, they focus on quick response to the problem, through prompt response and evaluation of claims as well as exemplary dispute review services. However, there is no mention of using good mechanism for detecting and responding to conflicts during planning and design. As the case studies established, there cannot be enough preparation during the process. Parts of the projects will change, specialists will change their mind; thus, there is need a for a way of providing effective detection and resolution of conflicts. This is the focus of this research.

Favela [Favela, 1993] also refers to studies made by Souder [Souder, 1988] which implies that there is a strong correlation between new products and "harmony." For Souder, harmony implies the ease with which all the participants in the process coordinate and communicate their work. This concept of harmony is relevant because it refers to the "how" of the communication and coordination. This research focuses on the "what" of the coordination and communication. This research holds the belief that this "what" is the design rationale.

## 2.4 Conflict Mitigation System Requirements

After the review of the issues involved in the design process, a set of requirements for the design rationale capture in a collaborative environment was developed. These requirements are based on how designers generate and retrieve information for supporting their design activities.

### 2.4.1 Representation and Management

A design rationale system should represent and manage the following:

- **Design intent evolution**

  Designers start the design with a set of requirements or design intents that they need to satisfy. These design intents rarely are expressed in the final design, except for the particular functions that each artifact performs. In other words, the design intents that tie all the artifact functions together are lost. Information that is necessary for effective conflict mitigation at later stages of the design is not available. In addition, design intents are not static or known at the beginning, they evolve as decisions are made during the process. New design intents are introduced or old intents change priority when designers make decisions. Some intents are introduced by the artifacts selected by the designers (e.g., arch and suspension bridges may introduce a totally different set of intents). In the case of priority and implicit intents, discussed in Section 2.2.4, when the structural engineer decided to use timber then the issue of durability surfaced explicitly. However, had the designer selected precast concrete, that issue would have low priority since it was expressed explicitly. By capturing the process that makes design intent evolution possible, designers can capture the initial design intents, the design intents generated through the design, and the plans which determine how these design intents are achieved.

- **Artifact evolution**

  Designers define artifacts at different levels of abstraction. This evolution combines two concepts: decomposition and refinement. Decomposition refers to the hierarchical disaggregation of an artifact. Refinement refers to a particular part of the artifact obtaining increasing detail in its structure, function, and behavior. In both cases the product starts with a structure, a set of functions, and a set of behaviors. These are expanded or refined as the design proceeds. In the case of decomposition, each of the lower level products performs a sub set of the functions that the overall product should perform. In the case of refinement, the most detailed product embodies all the functions of the earlier products.

- **Relationships: Intent-Intent and Intent-Artifact.**

  When the designers perform a particular task in the process, they use facts, assumptions, and different kinds of knowledge (e.g., heuristics and principles) to draw relationships between design intents. This knowledge explains that by accomplishing design intent A, design intent B is achieved to a certain extent. Such knowledge becomes important in explaining why a particular design intent was undertaken. In the same vein, designers use facts, assumptions, and different kinds of knowledge to select the artifact that satisfies the design intents. In addition, the designers often describe an artifact as satisfying a design intent because it overcomes the shortcomings of previously evaluated design alternatives.

### 2.4.2 Active Computer Support

The design process is not performed by one designer but between several designers who must interact and get feedback from each other. Not only is it necessary to represent and manage design intent evolution, artifact evolution and relationships, but it is also important to provide active computer support for negotiation between multiple participants and for the capture of design rationale. Active computer support is needed in two areas:

- **Negotiation between multiple designers.**

  In a collaborative environment, designers cooperate and negotiate in defining a product. This cooperation and negotiation implies changes in the design intents, artifacts, and in their relationships. The original designers' rationale serve as a basis for supporting the design; however, other designers may change that rationale by suggesting that new concepts to be taken in consideration.

- **Record and use of design rationale.**

  During the interviews performed for the case studies, the designers admit the value of recording the design rationale. However, they expressed concern about the time it will take them to record the design rationale. Thus, the computer has to obtain the information about the reasoning process of the designers without disruptions to them, i.e., in a non-intrusive manner. The computer needs to assume an active role in the recording process. Through use of different kinds of knowledge (e.g, heuristics, cases, catalog information), the computer can infer reasons for choices and suggest alternate designs.

## 2.5 Summary

From the case studies and additional literature review, it is concluded that the design/construction process can be defined as an collaborative-iterative decision-making activity organized to conceive the idea for, to prepare the description of, and to produce the plans by which resources are converted into artifacts or devices to meet societal needs. In addition, designers need to justify their decisions during the design/construction process ensuring prompt detection and mitigation of conflicts. Based on these ideas, this chapter presents the views that: 1) the designers' perspectives are expressed in their design rationale; 2) a system for capturing the design rationale needs to represent and manage design intent evolution, artifact evolution, and relationships between intents and between intent and artifact; 3) a design rationale system needs to capture its information in a non-intrusive manner by providing some of the design rationale; and 4) a system for conflict mitigation needs to provide active computer support for the negotiation between multiple participants.

# Chapter 3

# Representational Background

*The particular situation in which knowledge is used should influence the representation chosen.*

Morris W. Firebaugh, *[Firebaugh, 1989]*

## 3.1 Introduction

The main objective of this thesis is to explore how the computer can support conflict mitigation based using design rationale, which involves various specialists' objectives and reasons for rejecting or accepting a given alternative. However, design encompasses a broad range of activities that cannot be tackled by one single tool but by a collection of tools that target specific areas or task of the process. Thus, SHARED-DRIMS is not a stand alone tool. It is one of the computer aided tools for cooperative product development, collectively called DICE (Distributed and Integrated environment for Computer-aided Engineering). These tools all follow a set of guidelines that enable interaction and cooperation with the DICE framework. DICE is based on recent computer technologies such as knowledge-based systems, object-oriented methodology and database management systems, distributed processing, as well as networking which facilitate communication and collaboration. The following sections present the DICE framework and the reasons of its existence as well as the different methodologies that are involved in its modeling of information. Some of the sections are based on the work presented in [Wong and Sriram, 1993a] and [Gorti et al., 1993]. Section 3.2 provides an overview of the DICE framework. The explanation of the framework focuses on its architecture and organizational views. Section 3.3 explains the methodology for information modeling combining object-oriented methodology and its use in the representation of semantic networks. Finally, Section 3.4 provides a summary of all the ideas and work presented in this chapter.

## 3.2 DICE

### 3.2.1 Problems in the US Industries

The highly fragmented nature of product development in the US has caused various productivity problems, mostly resulting from the lack of communication and coordination. This poses several problems, as expounded by the following clip the April 30, 1990 issue of Business Week, p. 111 (see Figure 3-1 for a typical scenario in the AEC industry).

> "The present method of product development is like a relay race. The research or marketing department comes up with a product idea and hands it off to design. Design engineers craft a blueprint and a hand-built prototype. Then, they throw the design "over the wall" to manufacturing, where production engineers struggle to bring the blueprint to life. Often this proves so daunting that the blueprint has to be kicked back for revision, and the relay must be run again - and this can happen over and over. Once everything seems set, the purchasing department calls for bids on the necessary materials, parts, and factory equipment - stuff that can take months or even years to get. Worst of

all, a design glitch may turn up after all these wheels are in motion. Then, everything grinds to a halt until yet another so-called engineering change order is made."

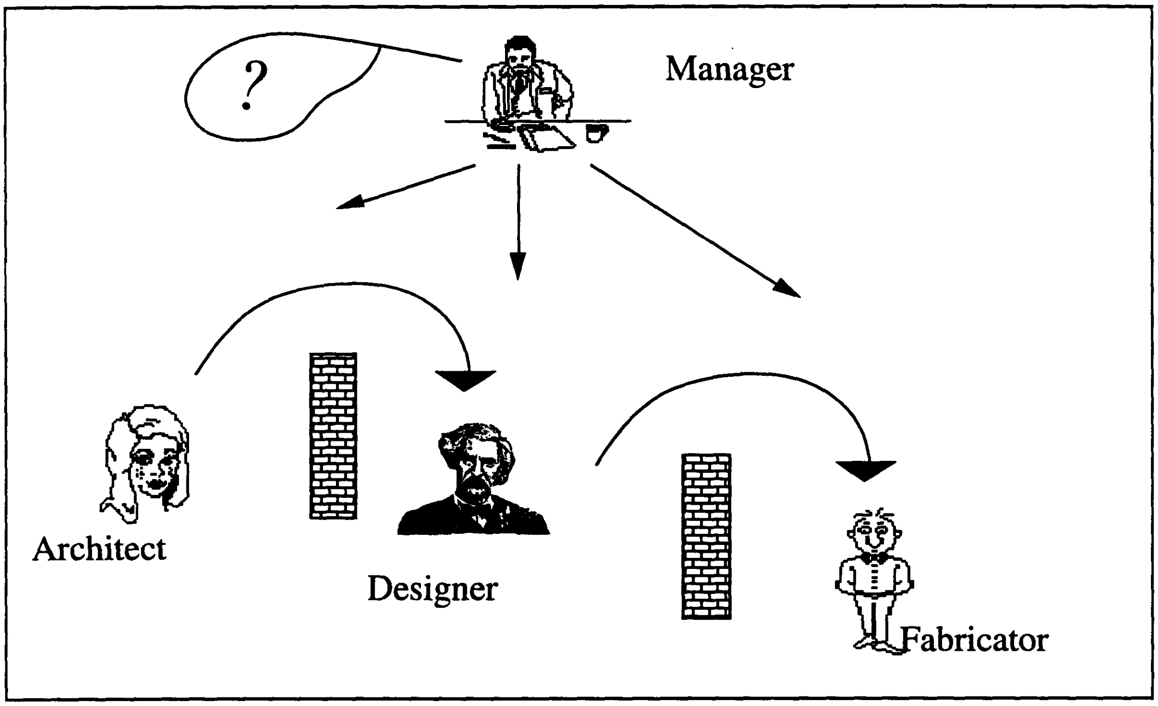


Figure 3-1: Over the wall engineering

Such problems routinely arise in the construction industry and cause several undesirable effects: 1) The construction process is slowed down, work stops when a conflict is found; 2) Prefabrication opportunities are limited, because details must remain flexible; 3) Opportunities for automation are limited, because expensive high speed equipment is incompatible with work interruptions from conflicts recognized in the field; 4) Rework is rampant, because field conflicts often require design changes; and 5) Conservatism pervades design, because designers provide excessive slack in component interfaces to avoid conflict. All of these problems decrease productivity.

### 3.2.2 Computer-Based Solution

With the current cost trends in computer hardware, it is likely that every engineer will have access to a high performance engineering workstation in the near future. Collaboration can then be facilitated by a network of computers/users providing a *virtual shared workspace*, as shown in Figure 3-2; the term *agent* is used to denote the combination of a human user and a computer.

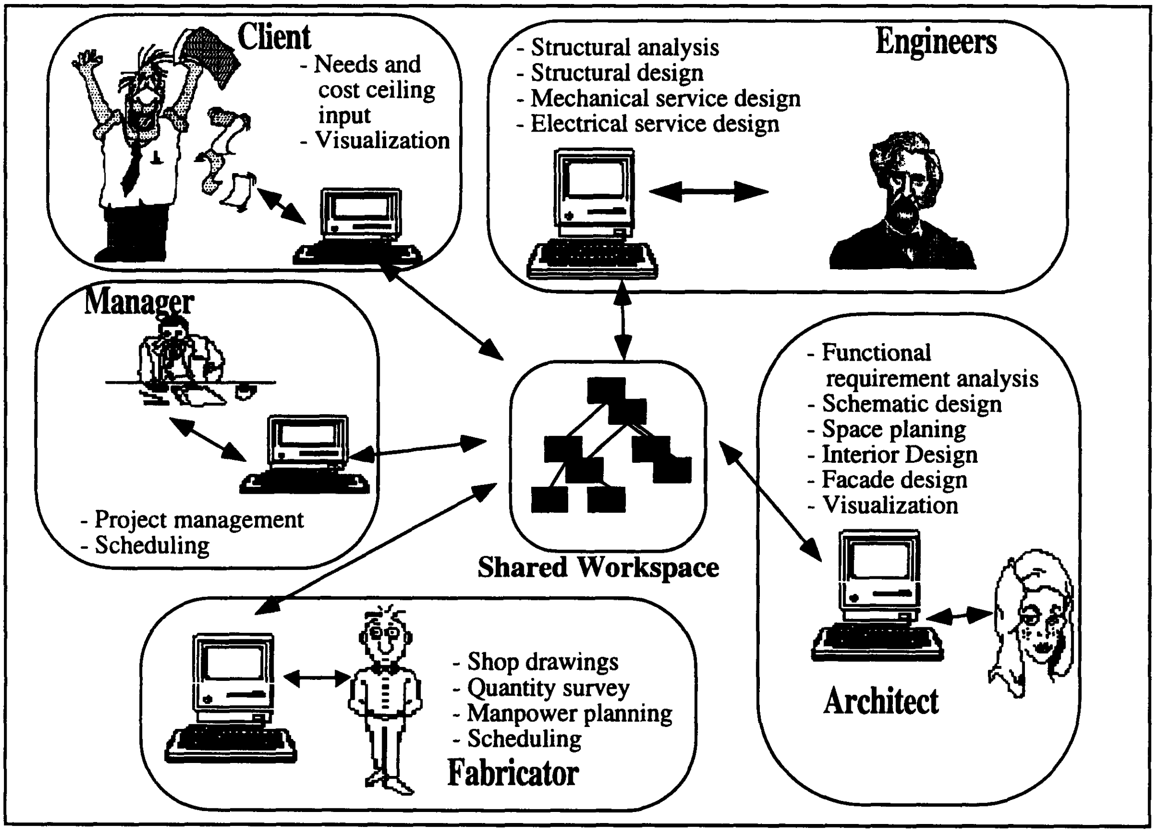


Figure 3-2: Computer-based view of cooperative product development

This is the philosophy taken in the DICE (Distributed and Integrated environment for Computer-aided Engineering) approach [Sriram and Logcher, 1993], where computer aided tools for cooperative product development are being developed to address the following objectives: 1) facilitate effective coordination and communication in various disciplines involved in engineering; 2) capture the process and rationale followed by which individual designers make decisions, that is, what information was used, how it was used and what it created; 3) forecast the impact of design decisions on manufacturing or construction; 4) provide designers with detailed manufacturing process or construction planning; and 5) develop a few design agents to illustrate the approach.

### 3.2.3 DICE Architecture

The DICE system architecture was developed based on current trends in advanced computing technology such as programming methodologies, object-oriented databases, organizational theory, graphical user interfaces, and knowledge based systems [Sriram et al., 1989]. DICE can be envisioned as a network of computers and users, where the communication and coordination is achieved through a global database and a distributed control mechanism. The components of DICE are described below.

**Blackboard**

The Blackboard is the medium through which communication takes place. The Blackboard (BB) in DICE is divided into three partitions: Solution (SBB), Negotiation (NBB), and Coordination (CBB). The Solution partition contains the design and construction information generated by various Knowledge Modules. The Negotiation partition contains negotiation traces between various engineers taking part in the design and manufacturing (construction) process. The Coordination partition contains the information needed for the coordination of various Knowledge Modules. In our current framework, the Blackboard is implemented over an object–oriented database management system (OODBMS).

**Knowledge Module/Agent**

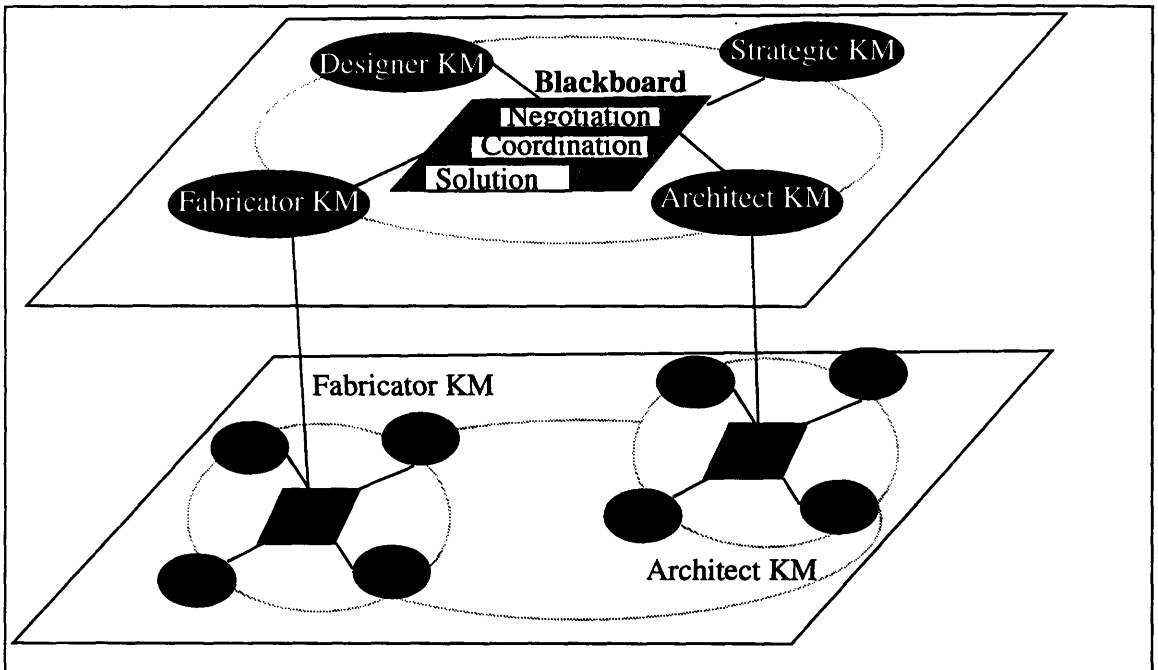
Each Knowledge Module (KM) or an agent can be viewed either as a knowledge based expert system (KBES) which carries out design and construction related tasks, a CAD tool, such as a database or an analysis program, etc., a user, or a combination of the above.

**Control Mechanism**

The Control Mechanism performs two tasks: 1) evaluate and propagate implications of actions taken by a particular KM; and 2) assist in the negotiation process. These are achieved through the object–oriented nature of the Blackboard and a Strategic KM. One major and unique difference between DICE and other Blackboard systems is that DICE's Blackboard is more than a static repository of data. It is an intelligent active database, with objects responding to different types of messages. A substantial part of the Control Mechanism's functionality is distributed to and localized in these active objects. In DICE's framework, any of the KMs can make changes to or request information from the Blackboard; requests are logged with the objects, and changes to the Blackboard may initiate either of two actions: finding the implications and notifying various KMs, or entering into a negotiation process, if two or more KMs suggest conflicting changes.

### 3.2.4 Organizational View of DICE

An organizational view of the DICE architecture is shown in Figure 3-3. This view is based on the work of Moses [Moses, 1987]. Two modes of communication are envisioned in carrying out a collaborative engineering process: formal communication through DICE's Blackboard and informal communication between Knowledge Modules (agents). The formal communication mode includes: 1) the creation, modification, and retrieval of objects in the Blackboard by the Knowledge Modules; and 2) message passing between the objects for system dictated coordination (e.g., consistency maintenance) and negotiation. The informal communication is direct communication between the Knowledge Modules (e.g., a request by an agent to clarify some details).



Solid lines indicate formal modes of communication
Dotted lines indicate infromal modes of communication

Figure 3-3: Organizational view of DICE

This report focuses mainly on the representation in DICE's Blackboard, specifically the solution partition. We also describe a partial framework for the control mechanism and communication facilities that supports the two modes of communication described above.

## 3.3 Information Modeling

The SHARED object model is now presented and used for the modeling of the information relevant to design rationale. This model combines some of the basic properties of object-oriented methodology and semantic networks, as explained in Section 3.3.3. In order to understand such a model and understand the different tools used for its implementation within this research, the following sections describe the object-oriented methodology, semantic networks, and the SHARED object model. The model is implemented with object-oriented programming; thus, one needs to know its characteristics and power. The semantic networks are used for describing the inter-object relationships in object-oriented programming. This network forms the basis for the model.

### 3.3.1 Object-Oriented Methodology

Object-oriented methodology is a new way of thinking about problems using models that describe real-worlds concepts. The fundamental construct is the object, which is an entity that combines both data structure and behavior. Object-oriented models are helpful for describing problems, communicating with experts, modeling enterprises, preparing documentation, and designing programs and databases.

In terms of programming this methodology, traditional programming, sometimes referred to as action-centered programming, (i.e., structured programming) maintains clear distinction between algorithms designed to operate on data and the data itself. However, this distinction set aside the possibility that different states may exist for data that affect how the data will be used. In other words, information about a problem domain is not always distinct from the rules for its interpretation and processing. Object-oriented programming is a paradigm with a different view of algorithms and data. Concepts, rules, and information about the use of data are incorporated within data itself using object-oriented programming. Another disadvantage of traditional programming is the extensive modifications required in the code whenever an addition is made in the data structure [Winston and Horn, 1984]. In object-oriented programming, objects may be modified without requiring users of the objects to change their code if the modifications are properly done.

The basic components of object-oriented methodology, are [Stefik and Bobrow, 1986]:

**Objects** are the organizational units of processing. They can be defined as individual, identifiable items, representing real or abstract entities, with crisp boundaries and meaning for the problem at hand. Objects combine state information and clearly defined protocol for describing the behavior of the represented entities.

**Classes** represent sets of objects with similar properties (state information) and common behavior. Each object is said to be an instance of its class. Each instance of the class has its own value for each attribute, but shares the attribute names and operations with other instances of the class.

**Messages** are the specification of actions to be performed by an object. Messages request objects to perform certain actions without specifying how the objects should perform it. The objects are free to execute the actions according to the objects' state.

**Methods** define the behavior of an object by telling it how it should respond to a message. The methods are the functions responsible for carrying the action requested in the message without allowing the message to have direct access to the object's internal structure.

Object-oriented methodology has some essential properties that allow complex systems to be formulated as modular programs. These properties include:

**Data abstraction,** which is the property of denoting the essential characteristics according to a purpose. Abstraction enables the definition of crisply conceptual boundaries of objects. By means of abstraction, objects can assume certain behavior according to the purpose that they are serving. Message passing and methods support the abstraction property. Messages request an action from an object for certain purpose while the methods use the relevant characteristics for that purpose in executing the action. Thus, an object may serve different purposes since many different abstractions of the same objects are possible.

**Data encapsulation,** which is the process of hiding all the characteristics of an object that are not essentials for the object purpose. This definition implies that encapsulation and abstraction are complementary concepts. Abstraction focuses on the outside view of the object while encapsulation prevents access to characteristics irrelevant for that view. Encapsulation is achieved by defining methods and messages as the interface of the object with its environment. State data is kept secret in the object and only the methods according to the behavior expected use the state data. Encapsulation separates implementation from abstraction. Thus, other objects do not need to know the implementation details of the characteristic of an object. They only need to know the behavior expected from the object. In this manner, encapsulation offers barriers among different abstraction of the same object.

**Inheritance** is the process by which an object obtains its structure and behavior from other objects. In the object-oriented paradigm this is done by the use of a hierarchical inheritance. Classes, as were defined, may represent objects of common structure and behavior; thus, the common structure and behavior of those objects can be defined by the class that represents them. And by inheritance, other objects can obtain these characteristics from the class.

Inheritance can be single and multiple. In single inheritance, objects only obtain their characteristics from one class while in multiple inheritance, the characteristics can be obtained from more than one class. In both cases, the hierarchical inheritance structure is composed of classes, subclasses, and instances. Classes can define a general structure or behavior of a group of objects. Subclasses can define a more specialized behavior of an object group. Instances are the lowest level of the hierarchical inheritance structure. An instance defines the structure, behavior, and characteristics (i.e., the state) of a given entity. Thus, instances cannot pass their characteristics to other instances through inheritance.

**Polymorphism** is the capability of different classes of objects to respond to the same set of messages in different ways. Polymorphism allows programs to treat uniformly objects that arise from different classes and respond to the same protocols.

Among the benefits that can be obtained by the use of object-oriented programming are:

- *Reusability*: Object-oriented programming style enhances and facilitates the reuse of code within a project and on new projects. Since objects encapsulate their structure and only provide an interface for communicating with other objects, the objects can be reused by different applications and only need to build interfaces for the new applications –if these interfaces do not exist. The reuse of software reduces the cost of design, coding, and testing by amortizing effort over several projects. In addition, code reuse within a project produces smaller programs and faster debugging.

- *Extensibility* is also achieved by the encapsulation principle of object-oriented programming. Software can be latter expanded by including new classes and interaction methods without the need to recreate or modify the existing code.

- *Robustness*: The use of abstraction and encapsulation may provide the programmer with good tools for detecting errors at run time. An object may support a well defined abstraction. Thus, the programmer determines the different functions that can be executed by the object within that abstraction, limiting the sources of errors.

### Object–Oriented Database Management Systems (OODBMS)

An object–oriented database management system basically provides database facilities together with an object–oriented data model for the definition and manipulation of data stored in its database. The database facilities include [Ahmed et al., 1992]:

- **Persistence**: the data resides in persistence storage, rather than in volatile memory, and can be used across sessions;
- **Concurrency**: multiple users access and use the same database simultaneously;
- **Transaction management**: a process which monitors database interactions to ensure consistency or correctness and stability of the data ;
- **Recovery**: the ability to recover from a crash to some defined stable state;
- **Query language**: a high-level, easy-to-use language for accessing information systematically;
- **Performance**: efficient access structures and algorithms for retrieving large amounts of persistent data from secondary storage; and
- **Security**: protection of information from unauthorized access.
- **Behavior**: the ability to combine the intended use of the data together with the data. This property allows for the objects to have behavior according to the context in which they are used.

Other data management facilities could also be provided by OODBMS. These facilities include:

- Version management [Kim and Chou, 1988];
- Composite objects [Kim et al., 1989]; and
- Schema (class) evolution [Banerjee et al., 1987].

A number of commercial OODBMS are currently available including Itasca$^{tm}$, Versant$^{tm}$, Ontos$^{tm}$, Objectstore$^{tm}$, $O_2{}^{tm}$, and Gemstone$^{tm}$. A survey of commercial OODBMS is provided in [Ahmed et al., 1992]. University based systems include Encore [Elmore et al., 1989], OSBT [Casais et al., 1992], and Exodus [Carey et al., 1989]. OODBMS, by integrating the object–oriented methodology with database facilities, provide a powerful medium for representation, storage, and management of complex information. Compared to relational database systems, OODBMS represent a superior medium for storing and managing engineering information. The reader is referred to [Ahmed et al., 1992] for descriptions of advantages of OODBMS over tradition relational databases management systems for capturing engineering information.

### 3.3.2 Semantic Modeling Schema

Semantic networks provide a means of relating objects into inter-object structures. Semantic networks were developed as a way of representing a psychological model of human associative memory [Quillian, 1968, Raphael, 1968]. They are useful for describing the relationship between objects and for drawing conclusions about their role and state [Firebaugh, 1989, Winston, 1984]. In a conventional object-oriented system, the semantic network is implemented by storing in named fields in the objects pointers to other objects. These named links are manipulated by procedures or methods in the object in order to give some operational meaning to the inter-object structure. These links and methods allow an object to participate in several inter-object structures at once without losing its identity. However, relationships need to be explicit so that they can be operated on and manipulated. Semantic networks have developed to allow for such representation. On of the most known models is the E-R model [Chen, 1976].

Some useful examples of inter-object structures used in decision rationale representation with their behavior are:

- Specialization/Generalization Hierarchies - representing classification systems. This relationship allows for the inheritance of structure and behavior from parent classes.
- Part-Of hierarchies, representing the decomposition of an entity into sub-entities. This relationship allows for selective inheritance. An object can inherit a behavior from a class, such as the color of walls of a house, while avoiding the inheritance of behavior that is restricted of the house, such as the shape of the house.
- Justification links, representing the justifications to recommendations presented by designers. This relationship builds the context in which the design is performed and verified.

To illustrate an entity participation in several inter-object structure, consider an *intent* object that might be the basis for a proposal, see Figure 3-4. The *intent* object might be:

- *introduced* by a recommendation,
- *composed-of* other *intents*,
- *refers-to* a *goal*.

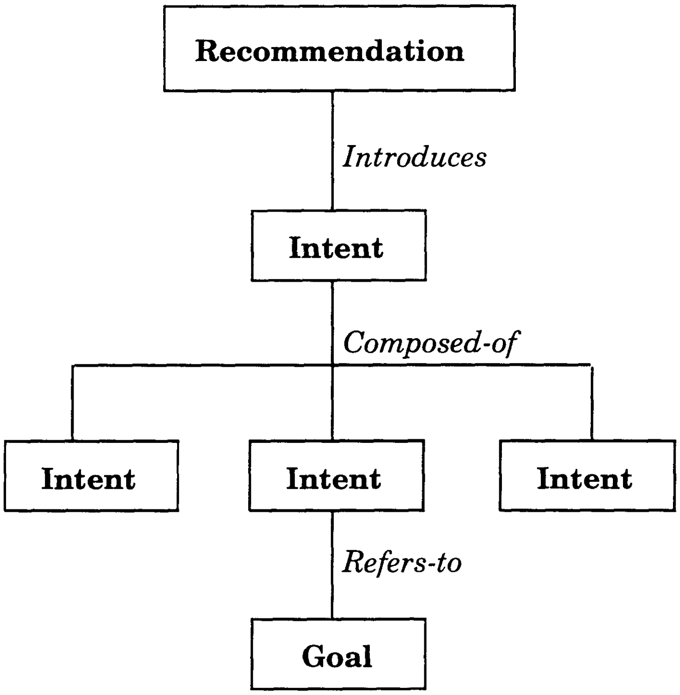


Figure 3-4: A Semantic Network.

The *intent* would participate in specialization, contains, and justification links by virtue of *specializes*, *composed-of*, and *basis-of* links respectively. The use of semantic networks in the representation of decision rationale is found in Chapter 5.

### 3.3.3 SHARED Object Model

In this section, we present the object model which forms the basis for the design knowledge representation. Our model is based on the SHARED object model, defined in [Wong, 1993], [Wong and Sriram, 1993a]. The SHARED model essentially extends the object-oriented methodology in the following manner [Wong, 1993], [Wong and Sriram, 1993a] :

1. *It provides explicit relationship entities with associated semantics and constraints,* instead of just using attribute references to objects. These relationships are associated with relationship classes and can be arranged in inheritance hierarchies as with object classes;
2. *It associates constraints with objects and relationships.* Constraints are used to maintain the consistency and integrity of a product model; and
3. *It provides a mechanism for handling the concept of "similar objects".*

We provide a brief overview of the SHARED object model to serve as the basis for further discussion.

#### Definition of Objects

A SHARED **object**, **o**, is defined as a *unique, identifiable entity* in the following form:

**Definition:**

$$\mathbf{o} = (\mathbf{uid}, \mathbf{oid}, \mathbf{A}, \mathbf{M}, \mathbf{R}, \mathbf{C}) \tag{3.1}$$

- **uid** is the unique identifier of an object. The set of all unique object identifiers is **UID**;

- **oid** is a non–unique similar object identifier related to the version of the object. It is used to refer to one of a set of similar objects which can be used to replace each other in relationships. Typically, we use this concept to model *alternatives* or *versions* of objects. Note that all versions of an object must be instantiations of the same class, whereas alternatives could represent any class. The set of all **oids** is **OID**.

- $\mathbf{A} = \{(\mathbf{t}_i, \mathbf{a}_i, \mathbf{v}_i)\}^*$. Each $\mathbf{a}_i$ is called an *attribute* of **o** and is represented by a *symbol* which is unique in **A**. Associated with each attribute is its *type*, $\mathbf{t}_i$. Each $\mathbf{t}_i$ has an associated domain, **domain(t**$_i$**)** = $\{\mathbf{v}_i\}$. Then, for ($\mathbf{t}_i$,$\mathbf{a}_i$,$\mathbf{v}_i$), $\mathbf{v}_i$ is called the *value* of $\mathbf{a}_i$ and $\mathbf{v}_i \in$ (**domain(t**$_i$**)** $\cup$ **nil**). If $\mathbf{v}_i$ = **nil**, ($\mathbf{t}_i$,$\mathbf{a}_i$,$\mathbf{v}_i$) can be written as ($\mathbf{t}_i$,$\mathbf{a}_i$). **A** can also have meta-attributes, which have a similar connotation to the attributes. i.e., each $\mathbf{a}_i = \{(\mathbf{t}_i, \mathbf{ma}_i, \mathbf{v}_i)\}$, where $\mathbf{ma}_i$ is a meta-attribute.

- $\mathbf{M} = \{(\mathbf{m_i}, \mathbf{tc_1}, \mathbf{tc_2}, ...\mathbf{tc}_n, \mathbf{tc})\}^*$

  Each element of **M** is a *method signature* which uniquely identifies a method. $\mathbf{m}_i$ is the method name represented by a symbol and $\mathbf{tc}_i$ is a *type*. The returned type of the method which can be a single valued or object type is specified by the last element in the tuple and the other elements define the types of the arguments of the method. Methods define operations on objects and have associated code. A method is defined as (*method signature, code*).

- $\mathbf{R} = \{\mathbf{rid}\}$, where **rid** is an identifier for a relationship. Relationships are discussed in Section3.3.3.

- $\mathrm{C} = \{\mathbf{cname}\}$. Each **cname** is a unique identifier for a constraint, **c** defined as (**cname**, **code**). A constraint can be viewed as $\mathbf{cname}() - > TRUE|FALSE$, that is a function which returns either *TRUE* or *FALSE*. Constraints may be used to restrict ranges of attributes, to define complex expressions on object attributes through rules,.etc.

For example, an object is defined as follows: [1]

$$(\mathbf{uid1}, \mathbf{oid2}, \{(\mathbf{int}, \mathbf{a}, 10), (\mathbf{String}, \mathbf{b}, \text{``}abc''\text{)}\}, \{(\mathbf{get\_a}, \mathbf{int}, \mathbf{int})\}, \{\mathbf{r1}, \mathbf{r2}\},$$
$$\{(\mathbf{c1}, a < 20)\})$$

where **uidl** is the unique identifier, **oid2** is a non-unique object identifier related to the version of the object, **int** and **String** are primitive data types, **rl** and **r2** are relationship identifiers. **cl** is a constraint on the value of the attribute **a**.

### Relationships

The SHARED model represents relationships between objects as objects themselves, thus making their semantics explicit. In particular, the relationships defined include **composition, functional** and **spatial** relationships, **version-of, alternative, sub-function, satisfied-by** and **requires** [Wong, 1993]. We now define a generic SHARED relationship as follows:

**Definition:**

$$\mathbf{r} = (\mathbf{rid}, \mathbf{RO}, \mathbf{A}, \mathbf{M}, \mathbf{C}) \tag{3.2}$$

where

[1] Object, relationships, method names, types are denoted by boldface fonts.

- **rid** is a unique identifier of the relationship **r**. The set of all unique relationship identifiers is **RID**.
- RO = $\{(\mathbf{t}, \mathbf{ro}, \mathbf{v})\}$.

  Each **ro** $\in$ **RO** is called a *role* of a relationship. **ro** is the role name of a role and **v** is the *value* of a role: a wellformedness condition is that **v** $\in$ {**OID** $\cup$ **UID**} or **v** $\subset$ {**OID** $\cup$ **UID**}, and **v** $\in$ domain(t) where **OID** is the set of all object identifiers and **UID**, the set of all unique object identifiers, and **t** is a type. Furthermore, there must be at least two objects partaking in the roles of a relationship. For a relationship among a particular set of objects to be valid, each of the objects must be identified by some role in the relationship and each of the objects must include the particular relationship in the relationship set **R** of the object's definition.
- **A** is a set of attributes of a relationship, defined in a manner similar to **A** of an object;
- **M** is a set of methods, defined in a manner similar to **M** of an object. The methods define operations on the roles and attributes of the relationships; and
- **C** is a set of constraints on objects associated with the roles of the relationship and its attributes (interaction constraints). It includes constraints on cardinality of roles. It is defined in the same way as **C** of an object.

For example, a relationship could be defined as follows.

**(r1, {(System, composite, s1), (Set_System, subsystems, {s11, s12, s13}), (String, description, "a part of rel")},{(get_subsystems, Set_System)}, {c1})**

where **System** is a class, **Set_System** denotes a set of **Systems**, **s11**, **s12**, and **s13** are identifiers of objects which constitute this set and **c1** is a constraint. The method **get_subsystems** is the access function to return the subsystems, and does not take any arguments. However, other functions could take any number of arguments and perform complex and lengthy tasks.

**Classes** are defined on the objects and relationships defined above, as abstraction mechanisms to make the common properties and semantics explicit. For formal definitions of these mechanisms, we refer the reader to [Wong, 1993].

Now presenting the object and relationship classifications: A SHARED object, **o**, is classified as an *instance* of a class, **c**, if

- ∀ **ac** ∈ **c.A**, ∃ **a** in **o.A** such that **a** = **ac** or **a** is the same as **ac** except **a** is bounded to a value in **o** while value of **ac** is **nil**;
- ∀ **r** ∈ **o.R**, ∃ **cr** ∈ **c.R** such that **r** ∈ **domain(cr)**;
- ∀ **m** in **c.M**, ∃ **mc** ∈ **o.M** such that **m** = **mc**; and
- ∀ **con** in **c.CON**, ∃ **ccon** ∈ **o.CON** such that **con** = **ccon**.

Furthermore, since an object, **o** must be in one of the roles of all its relationships, the type (class) of the role in which **o** is in must be one of the class in which **o** is an instance of.

Generalization and specialization are also defined in terms of the class abstractions. These are relationships between classes which define a partial order on the set of all classes (i.e., they are *reflexive, antisymmetric*, and *transitive*). Generalization is used as an implementation mechanism for sharing code among more specialized classes. That is, a specialized class can inherit properties of a number of more general classes, in a process known as *multiple inheritance.*

### 3.3.4 Knowledge-Based Expert System (KBES)

Certain intelligence can be coded into the methods of objects in a procedural way. Examples of this are constraints which can be coded procedurally in the methods. KBES technology provides a more convenient and flexible mechanism for representing knowledge in the form of rules (e.g., constraints such as those in building codes).

A KBES provides a higher level programming tool compared to conventional programming environments [Sriram, 1988]. A KBES can be defined as a Computer program which incorporates knowledge and reasoning in solving difficult tasks usually performed by an expert.

A KBES can be considered as consisting of three basic components:

- *Knowledge Base* which is a collection of general facts and rules about the problem domain.
- *Inference Mechanism* which combines the facts and rules to deduce new facts. Different types of inference mechanism are available. Typical types are forward chaining, backward chaining, hierarchical refinement, etc. [Sriram, 1988].
- *Context* is the workspace for the solution constructed by the inference mechanism from the information provided by the user and knowledge base.

In an object–oriented framework, both the facts and the rules, and the context can be represented as objects. Similarly, the inference mechanism can be implemented as an object that has methods which are capable of performing the inferencing, given a set of fact objects and rule objects. This report incorporates such a framework for constraint declaration and checking.

## 3.4 Conclusions

The availability of environments such as DICE provides the foundation for work in the capture and use of design rationale. DICE covers most of the activities related to the design process such as source of knowledge, establishment of process for combining parts into wholes, as well as handling of qualitative and quantitative constraints. The interesting part of the DICE environments is that its components modules can produce the process by which they arrive to a solution. This allows the incorporation of that information to the design rationale.

Another topic discussed in this chapter is the use of object-oriented methodology to represent items in the real world. This methodology allows the modeling of entities that take different roles during their lifetime, as usually happens with real world entities. In addition, the chapter presents how semantic networks can be combined to object-oriented methodology to produce a more explicit model of the entities and relationships of objects in a given world. This combination led to the development of the SHARED object model which is being used to represent the constructs and primitives of the design rationale model.

# Chapter 4

# Related Research

*Progress, far from consisting in change, depends on retentiveness . . . Those who cannot remember the past are condemned to fulfil it.*

George Santayana, *Life of Reason, vol. I, ch. xii, 1905-6*

## 4.1 Chapter Introduction

Capturing rationale has been a research topic for several decades. There have been a number of models and systems developed by researchers in different application areas ranging from discourse, [Toulmin, 1958], to engineering design, [Garcia and Howard, 1992]. Figure 4-1 shows a classification of these research efforts according to the requirements presented in Section 2.4.

In Figure 4-1, the **Y** coordinate represents the number of designers that are able to record their interacting rationale and are able to participate in the mitigation of the conflicts. The scale is divided into *single* and *multiple participants*. In other words, this parameter represents how the different models or systems handle different designers inter-working

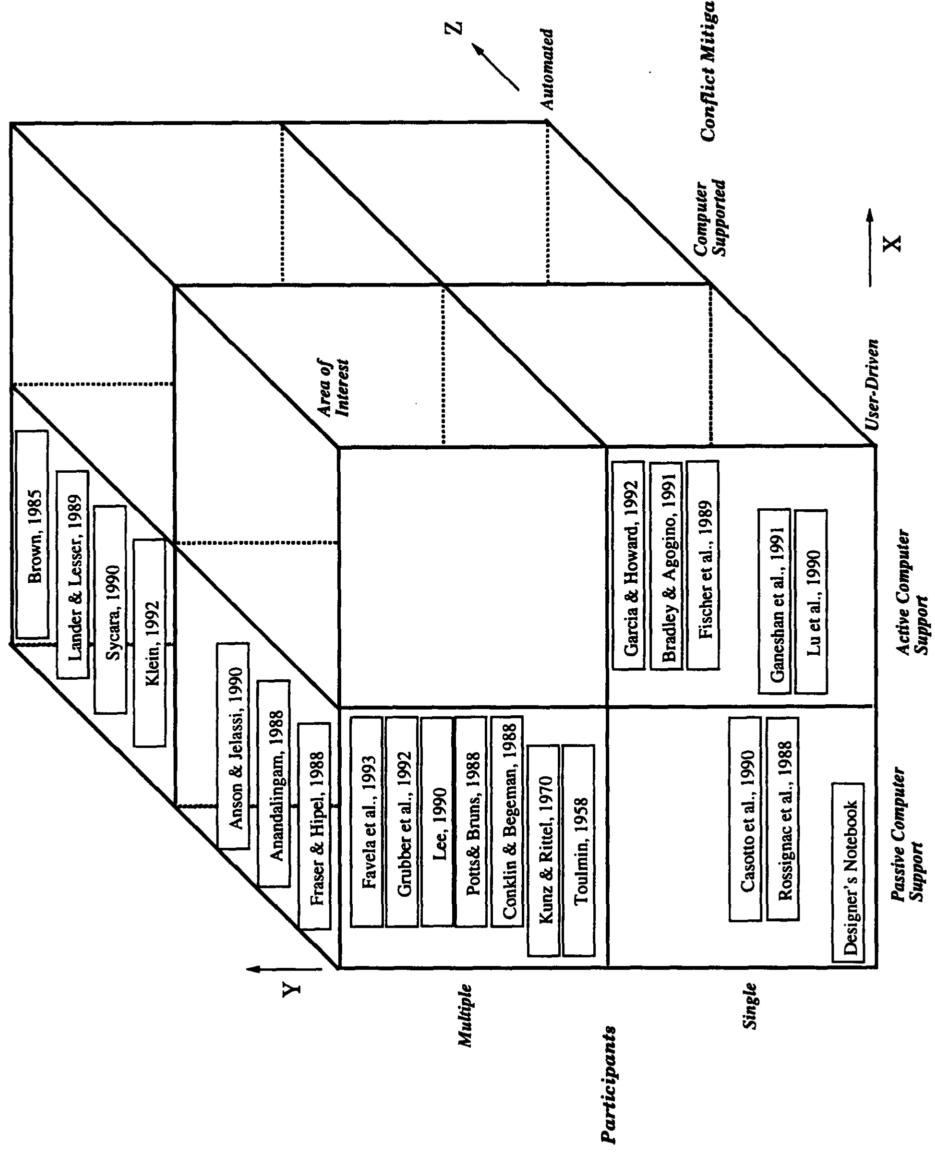


Figure 4-1: Comparison of Design Rationale research efforts.

and relating to each other on generating a product. The **X** coordinate represents the computer support for recording and using of the rationale for conflict mitigation. The scale is divided into *passive* and *active computer support*. *Passive computer support* indicates that the computer helps the designer to store the rationale. The designer inputs the rationale in the computer and the system creates some links between the different components of the rationale. *Active computer support* indicates that the computer helps in recording the rationale by providing part of it. The **Z** coordinate represents the support provided by the computer during conflict mitigation. The scale is divided into *user-driven*, *computer supported*, and *automated*. *User-driven* indicates that the user inputs most of the intents (preferences) and recommendations (options) into the system and the computer uses some general strategy like game and bargaining theories to evaluate recommendations with respect to the intents. *Computer supported* indicates that the computer provides some of the intents and recommendations to be analyzed, as well as it provides some domain dependent knowledge (i.e., heuristics, cases, first principles, etc.) for mitigating the conflicts. Of course, this does not preclude user interaction and application of general strategies, as available in *user-driven* systems. *Automated* indicates that the computer provides solutions to the conflict with very little interaction with the user, where intents and recommendations are implicit in the conflicts and the solutions presented.

The **X** scale in Figure 4-1 is a continuous measurement with more computer support as the boxes get farther away from the origin. The **Y** scale is discrete and there is no relation between the distances of the boxes to the origin. The **Z** scale is a continuous measurement ranging from mostly user-driven mitigation to mostly computer automated mitigation with a middle balance where an interactive user-computer mitigation is achieved.

The selection of these three parameters is due to the nature of design and the requirements of the designers. The systems have to support the negotiation between multiple designers; thus, they need to support conflict mitigation. In addition, designers want support in the recording and using design rationale. Most of the previous research efforts focused on the model of the rationale but there was little emphasis on the utilization of the rationale as discussed in the following sections. The representation requirements established in Section 2.4 deal more with the modeling of rationale and are useful for the development of the ontology. The requirements for active computer support deal more with the usability of the rationale and this is the focus of this research.

There is thus a gap in the *multiple participants-computer supported conflict mitigation-active computer support design rationale* quadrant. Little or no documentation of research in this area exists. However, this quadrant is presented by Section 2.4 as the quadrant where a system or model is necessary. Thus, a model and system for capturing the rationale of negotiating participants in which the computer provides support for providing rationale and mitigating the conflicts is necessary.

Section 4.2 offers a summary of related research in the area of design rationale representation. Section 4.3 provides an overview of related research in the area of conflict mitigation. In these sections, the relationship of this dissertation research with the previous work is also denoted to present where the research fits in the area of capturing design rationale and conflict mitigation. Finally, Section 4.4 provides a summary of this chapter.

## 4.2 Related Work on Design Rationale

Most of the research in design rationale has focused on capturing design rationale without concern for its later use. The use has been limited to maintaining the design history. In that case, the design rationale models or systems fall into the plane *participants-design rationale* without going into the *conflict mitigation* direction. Section 4.2.1 presents the research efforts on the *single participant-passive computer support models. Multiple participant-passive computer support models* are presented on Section 4.2.2. Finally, Section 4.2.3 provides a summary of the *single participant-active computer support models.*

### 4.2.1 Single Participant-Passive Computer Support Models

In Figure 4-1, the *single participant-passive computer support* quadrant has the designer's notebook which represents the notes taken by the designer during the design process. This document is usually private and manually developed. It also has Rossignac *et al.*'s MAMOUR [Rossignac et al., 1988] and Cassotto *et al.*'s VOV [Casotto et al., 1990] which keep a trace of the design as it evolves, but leaves the design intent implicit in the trace. The idea behind these systems is that a sequence of transformations represents the design and captures some of the designer's intent. Here, the transformations are operations performed on a model, and the sequence of these operations give the final product. Thus, it is believed that by recording that sequence, the product could be reproduced, if needed. One important point is that design rationale is defined as the operations that can re-create the product while intent is believed to be the operations performed. As explained in Section 2.4, intents are more than operations. They also refer to objectives to be achieved which are not related to a specific task but to the comparison between design alternatives.

### 4.2.2 Multiple Participants-Passive Computer Support Models

The *multiple participants-passive computer support* quadrant has a series of research efforts from academia and industry: Toulmin's Model [Toulmin, 1958]; Kunz and Rittel's Issue Based Information System (IBIS) [Kunz and Rittel, 1970]; Conklin and Begeman's

Graphical Issue Based Information System (gIBIS) [Conklin and Begeman, 1988]; Potts and Bruns' Model [Potts and Bruns, 1988]; Lee's Design Representation Language (DRL) [Lee, 1990]; and Grubber *et al.*'s SHADE [Grubber et al., 1992]; and Favela *et al.*'s CADS [Favela et al., 1993]. An important note in this quadrant is the ontology used by these systems. Their ontology lacks a representation and a structure for the process and the product as they evolve. Missing is the notion of artifact evolution. Most of them concentrate on the decisions made but without any underlying model of the artifact. Also missing is the notion of classification of the intents (i.e., objectives, constraints, function, and goals), as well as the classification of the justifications for a proposal (i.e., rules, catalog entry, first principles, etc) since they have different characteristics and are used different by the designers. Section 5.2 explains in more detail these classifications. In addition, these systems do not really attempt to perform any conflict mitigation. This is due to the lack of structure of the models. It will be difficult to assert that an intent can only be satisfied after comparison between different alternatives when there is no control mechanism to enforce that.

### 4.2.3 Single Participant-Active Computer Support Models

The *single participant-active computer support* quadrant has Thompson and Lu's AIDEMS [Thompson and Lu, 1990]; Ganeshan *et al.* system [Ganeshan et al., 1991]; Fisher *et al.*'s JANUS [Fischer et al., 1989]; Garcia and Howard's ADD [Garcia and Howard, 1992]; and Bradley and Agogino's DESIGN SCRIBE [Bradley and Agogino, 1991]. These systems capture design rationale from the perspective of a single designer. They lack support for multiple designers changing each other's design rationale due to intents outside of a particular design domain. Their models also lack the classification of intents and justifications as in the *passive computer support* models. These systems did overcome the lack of structure and representation of the product and process inherited in systems of the *passive computer support* quadrant. This enabled them to use information for inferencing and presenting part of the design rationale.

## 4.3 Related Work on Conflict Mitigation

Models or systems in the area of conflict mitigation have focused primarily on the resolution of conflicts. To that end, they have provided support in terms of evaluating participants' options (*user-driven* systems) or in terms of providing solution to the conflict based on some domain-dependent knowledge (*Automated* systems). However, they have lacked support in the area of rationale capture, conflict causes, and conflict prevention. In addition, a balance is needed in terms of *user-driven* and *automated* support. Some solutions will be available on domain-dependent knowledge (i.e., heuristics, rules, first principles, etc.). However, some novel solutions will come from the users/designers experience in dealing with similar problems. Thus, support needs to be provided such that both user resolution and computer solution can co-exist. Section 4.3.1 presents research in the area of *user-driven* support. Section 4.3.2 outlines research efforts in the area of *automated* support.

### 4.3.1 User-Driven

The *multiple participant-user driven- passive computer support* quadrant has a series of research efforts: Fraser and Hipel's Conflict Analysis [Fraser and Hipel, 1988]; Anandalingan and Apprey's use of bi-level linear programming [Anandalingam and Apprey, 1992]; and Anson and Jelassi's use of integrative bargaining [Anson and Jelassi, 1990]. These systems take designers' options and evaluate them helping the designers select the best option. However, the computer does not provide any support in generating some of these options and their accompanying preferences.

### 4.3.2 Automated

The *multiple participant-automated- passive computer support* quadrant has a series of research efforts: Brown's CYL [Brown, 1985]; Sycara's PERSUADER [Sycara, 1989]; Lander and Lesser's CEF [Lander and Lesser, 1989]; and Klein's DRCS [Klein, 1992] and Conflict Hierarchy [Klein et al., 1990]. The shortcomings of these systems are presented by their position on the graph. Sycara's and Klein's approaches get closer to the computer supported spectrum of conflict mitigation but still lack the active computer support for capturing the design rationale. These systems provide some of the designers options but the designers' preferences are implicit in the computer recommendations.

## 4.4 Summary

Research related to design rationale and conflict mitigation support was described in this chapter. The need for describing a more powerful design rationale model surge of the limitations of the existing models to describe the design problem relationship explicitly. In addition, these models do not represent all the elements that are relevant for the design rationale capture. Some models have limitations in their representation of intents while others have limitations in representing the design elements interactions. In the area of conflict mitigation, the need for a system that makes explicit the designers' options and preferences was shown.

# Chapter 5

# DRIM Information Model

*An engineering drawing contains the results of the decision-making process in a design, but does not record how and why the decisions were made.*

Ganeshan, Finger and Garrett, *[Ganeshan et al., 1991]*

## 5.1 Introduction

As stated in Section 2.3, design can be defined as an iterative decision-making activity to conceive an idea for, prepare the description of, and produce the plans for the process by which resources are converted into artifacts or devices to meet human needs. Design as a decision-making activity requires a problem-solving strategy. Section 2.3 presented the concept of opportunistic design [Banares-Alcantara, 1991] in which designers use *top-down* or *bottom-up* approaches inter-changeably. These approaches permit designers to tackle problems at the desired level of detail. However, some problems may arise in terms of consistency if the interactions among the sub-problems are not well organized and specified.

These problem-solving strategies also support the segmentary and collaborative nature of design[1]. Sub-problems can be mapped to the different disciplines involved in the design when a functional division is used. On the other hand, if a spatial division is used, sub-problems can incorporate all the disciplines. In either case, the difficulties in keeping track of the interactions among sub-problems are exacerbated. One needs to not only keep track of the sub-problems, but also to keep track of the interactions. For example, a discipline *A* in the design team may decompose and solve a design problem, and another discipline *B* might not know the assumptions and compromises made to achieve the decomposition or the solution. However, these assumptions or compromises made by discipline *A* might affect or become invalid on discipline *B*. If *B* can access the assumptions made by *A*, then *B* can prevent *A* from pursuing a decomposition unlikely to fix the problem from B's perspective. On the other hand, if *B* does not know the assumptions, then, in addition to discovering the interference later in the project (when it is more expensive to fix), *B* might not know the causes of the problems. Even when the design is produced by one discipline, the iterative nature of the design introduces a dynamic element to sub-problem interaction. These interactions may change during the design process. All these actions and information form part of the design rationale of the artifact.

For these reasons, the *top-down* or *bottom-up* approaches need to capture the design rationale behind their solution. Since these approaches fail when conflicts are detected, designers cannot follow a path up or down but they have to iterate. As discussed in Section 4.2, there is a gap in the *multiple participants-active computer support* quadrant. There is a need for a model that allows rationale information to be captured from multiple participants with active computer support. This thesis presents a model called DRIM which

[1] Segmentary nature means that multiple professionals from multiple disciplines participate in the design of an artifact.

stands for Design Recommendation-Intent Model. DRIM can represent the current method by which designers tackle problems and sub-problems when setting the characteristics of an artifact. During the design process, designers make recommendations about these characteristics. For example, a structural engineer recommends prestressed concrete as the material for a bridge. However, this recommendation only indicates what is to be done. It lacks information about who made that recommendation, what the designer was trying to accomplish, and why the designer believes that the recommendation will serve its purpose. DRIM facilitates the capture of the above information.

A detailed description of the components of the model is given in Section 5.2. Section 5.3 presents the definitions of the relationships between the different components in DRIM. Design as a process with the DRIM objects is explained in Section 5.4. Section 5.5 shows design rationale trace using DRIM components and relationships. Finally, Section 5.6 provides a summary of this chapter.

## 5.2 DRIM Primitive Classes

The DRIM model overcomes the deficiencies of the models presented in Section 4.2 by providing a method by which design rationale information from multiple participants can be partially generated, stored and later retrieved by a computer system. DRIM supports the natural way in which designers select the characteristics of an artifact. During the design process, designers make recommendations about these characteristics. For example, a structural engineer recommends prestressed concrete as the material for a bridge. However, this recommendation only presents what is to be done. They lack information about who made that recommendation, (i.e., a computer or a human), what the designer was trying to accomplish, and why the designer believes that the recommendation is correct.

The DRIM model overcomes the above deficiencies by providing mechanism by which that information can be stored and later retrieved. DRIM consists of a proposal which is related to the designer who presents it, and the intent that is sought. The proposal, in turn, is composed of the recommendation that satisfies the intent and the justifications for presenting such a recommendation. The various components of DRIM, shown in Figure 5-1, are discussed below; the graphical notation used is the Object Modeling Technique (OMT) [Rumbaugh et al., 1991] and the representational model is the SHARED model (explained briefly in Section 3.3.3 and found in more detail in [Wong and Sriram, 1993a]).

### 5.2.1 Designer

A designer represents the entity (human, e.g., a structural engineer, and computer, e.g., a synthesis program) that presents a proposal, based on a design intent that needs to be satisfied. Through the presentation of conflicting or supportive proposals, different designers enter into a negotiation process. Supportive proposals from other designers can be used as support arguments for a designer's proposal. In that case, the designers are not really negotiating but are collaborating.

A designer and its sub-classes are defined using the SHARED object model as a tuple of the form:

**(Designer, oid, A, M, R, C)**

where

**A** is a set composed of {(**String**, **speciality**), (**Address**, **office**)} where **speciality** is a character string representing the designer's profession. **office**

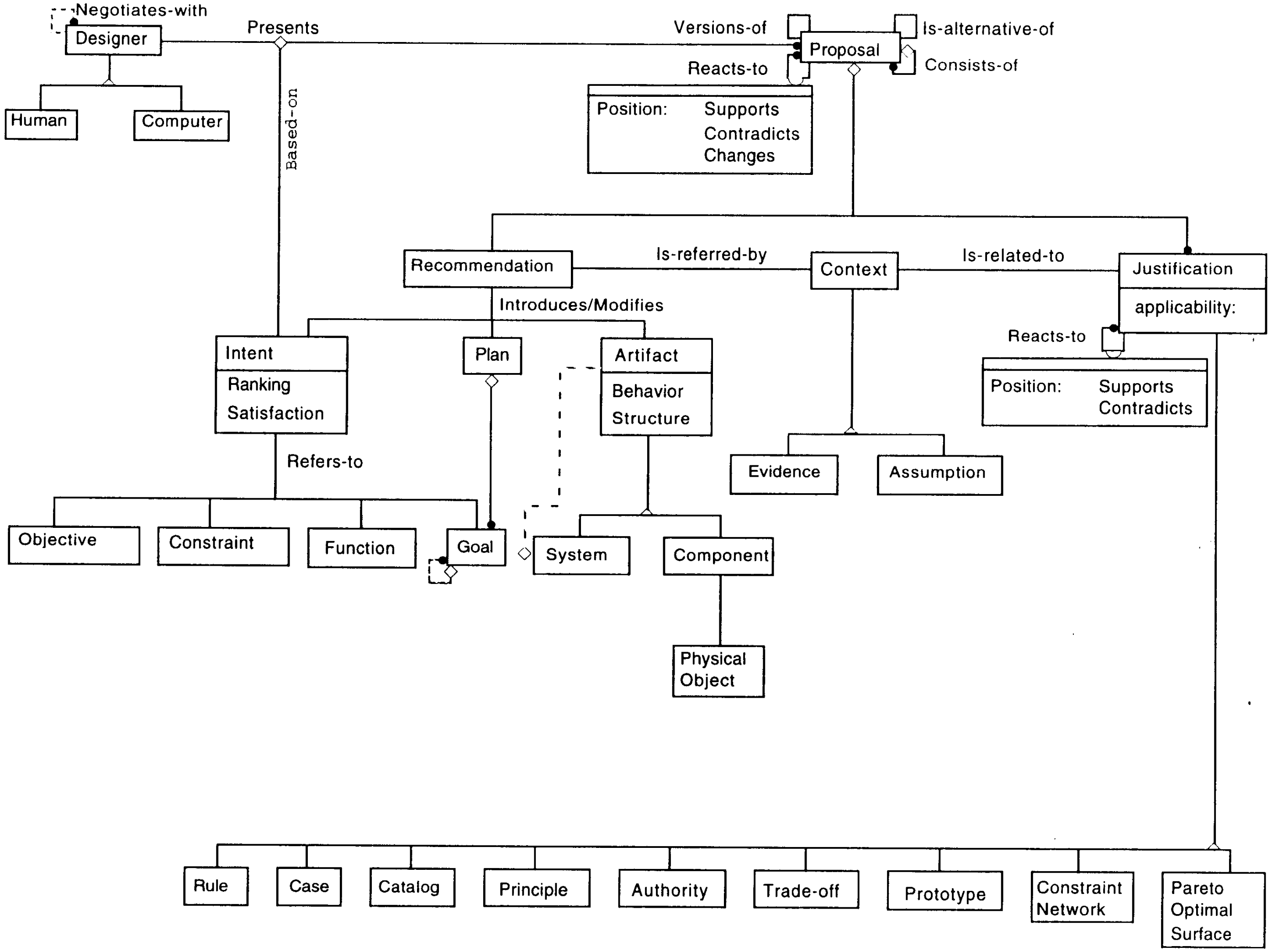


Figure 5-1: DRIMS components (the graphical notation used is the OMT model [Rumbaugh et al., 1991]).

is an object attribute of the **Address** class. This class has attributes for street, number, apartment, city, state and zip-code.

**M** is a set of methods related to the designer's information, presentation of proposals, as well as identification of negotiating designers and conflicting proposals.

**R** is a set composed of {**Negotiates-with, Presents**}. **Negotiates-with** links two designers that have conflicting or supporting proposals. **Presents** links the designer with presented proposal and the intent that needs to be satisfied.

**C** is a set of constraints which involve the expertise of the designer and the recommendations being made.

A human is defined as a tuple of the form:

**(Human, oid, A, M, R, Designer.C)**

where

**A** is a set composed of {**Designer.A, A'**}. **Designer.A** is the set of all the attributes from the **Designer** class. **A'** is a set composed of {**(String, e-mail)**}. **e-mail** is a character string representation of the designer's e-mail address.

**M** is a set composed of {**Designer.M, M'**}. **Designer.M** is the set of all the methods from the **Designer** class. **M'** is a set of methods related to the human's information.

**R** is a set composed of {**Designer.R, R'**}. **Designer.R** is the set of all the relationships from the **Designer** class. **R'** is a set composed of {**Supervisor**}. **Supervisor** links a designer with the supervising designer if there is one.

**Designer.C** is the set of all the constraints from the **Designer** class.

A computer is a software program that resides on a given platform. A computer has similar characteristics to a human and is defined as a tuple of the form:

**(Computer, oid, A, M, R, Designer.C)**

where

**A** is a set composed of {**Designer.A, A'**}. **Designer.A** is the set of all the attributes from the **Designer** class. **A'** is a set composed of {(**String, internet)**}. **internet** is a character string representation of the computer's internet address or its host name.

**M** is a set composed of {**Designer.M, M'**}. **Designer.M** is the set of all the methods from the **Designer** class. **M'** is a set of methods related to the computer's information.

**R** is a set composed of {**Designer.R, R'**}. **Designer.R** is the set of all the relationships from the **Designer** class. **R'** is a set composed of {**Co-designer**}. **Co-designer** links a computer with the cooperating human designer if there is one.

**Designer.C** is the set of all the constraints from the **Designer** class.

### 5.2.2 Proposal

A proposal represents the statement given by a designer. A proposal includes the recommendation believed to satisfy a design intent and the justifications of why the recommendation fulfills that design intent.

As designers present different proposals based on the same design intent, they create *is-alternative-of* relationships between the proposals. The *is-alternative-of* relationship is used to associate a proposal with another "similar" proposal which it can replace. Alternative proposals contain recommendations that are usually instances of different recommendation classes which satisfy the same intent and perhaps additional intents. For example, a structural engineer recommends the use of prestressed concrete for a bridge as an alternative to timber. This relationship type allows justifications to be retrieved from previously evaluated recommendations where the design alternatives have similar parts. When designers want to further increase the detail of a recommendation they create *versions-of* the original proposal, e.g., the proposal of using steel girders of 20 foot length for the bridge is a version of the proposal to use steel girders of 15 foot length for the same bridge. *Versions-of* links are the result of the designer's aim to satisfy new intents that were not taken in consideration before. The difference between *is-alternative-of* and *versions-of* links is the amount of change to the recommendation in order to satisfy the new intents. The *versions-of* link only presents incremental change on the recommendation, usually adding detail to the same concept in order to satisfy the new intents. On the other hand, the *is-alternative-of* link presents a new concept to satisfy the same intent – the change is more radical.

A proposal may *consist-of* sub-proposals. For example, the proposal for designing a bridge can consist of the sub-proposals related to the sub-goals of selecting the material and the structural form of the bridge. This relationship allows designers to decompose a proposal by presenting other proposals. These sub-proposals define either a process or a part hierarchy. During negotiation designers may present a proposal that *reacts-to* an already presented proposal. For example, the proposal to remove the central pier of the bridge reacts to the proposal of using a two span bridge with central support. This relationship allows designers to present different views, facilitating the negotiation process by making these views explicit.

A proposal is defined using the SHARED object model as a tuple of the form:

**(Proposal, oid, A, M, R, C)**

where

**A** is a set composed of {**(String, status)**, **(Recommendation, proposed-recommendation)**, **(Intent, sought-intent)**}. **status** is a character string representation of the proposal's status (either *in-process* or *finished*). **proposed-recommendation** is an object attribute of the **Recommendation** class which refers to the recommendation proposed by the designer. **sought-intent** is an object attribute of the **Intent** class which refers to the intent asserted by the designer.

**M** is a set of methods which define the behavior of proposals. These methods provide proposal's information retrieval and comparison, as well as mechanism for building the relationships between proposals (i.e, versions-of, consist-of, is-alternative-of, and reacts-to).

**R** is a set composed of {**Presents**, **Versions-of**, **Is-alternative-of**, **Reacts-to**, **Consists-of**}. **Presents** links a proposal with the designer who presents it. **Versions-of** links two proposals that introduce recommendations with incremental changes. **Is-alternative-of** links two proposals that introduce recommendations that are different but satisfy the same intent. **Reacts-to** links two proposals that introduce recommendations that contradict, or support each other. **Consists-of** links two proposals in which one is the parent proposal and the other is a sub-proposal.

**C** is a set of constraints which requires a recommendation and at least one justification to be present in order for a proposal to exist.

### 5.2.3 Intent

A design intent refers to what the designer wants to achieve or satisfy. A design intent has a ranking that specifies its importance and a satisfaction measure that specifies the extent to which the design intent has been satisfied by a recommendation. A design intent refers to the following attributes of a design: (1) objectives, (2) constraints, (3) functions, or (4) goals. The collection and hierarchy of the design intents together represent the design process. Definitions of objective, constraint, function, and goal are provided below. The relevance of this classification is that each one of these intents has different characteristics and the designers respond differently to these intents.

An intent is defined using the SHARED object model as a tuple of the form:

**(Intent, oid, A, M, R, C)**

where

**A** is a set composed of {**(Int, ranking), (Int, satisfaction)**}. **ranking** is an integer representation (from 0 to 100) of the intent's ranking. This attribute represents the priority of achieving the intent - the higher the number, the higher the priority. **satisfaction** is an integer representation of the intent's satisfaction. This attribute represents the degree (from 0 to 100) to which the designer believes that the intent has been satisfied.

**M** is a set of methods which define the behavior of intents. These methods provide intent's information retrieval and comparison, as well as mechanism for building the relationships between proposals and intents (i.e, based-on and percentage of satisfaction).

**R** is a set composed of {**Based-on**, **Introduces**, **Consists-of**, **Modifies**, **Refers-to**}. **Based-on** links an intent that needs to be satisfied, a designer that presents the proposal that satisfy the intent, and the proposal itself. **Introduces** links an intent with the recommendation that introduces it. **Consists-of** links an intent with its sub-intents. **Modifies** links an intent with the recommendation that modifies it. **Refers-to** links an intent and a need (objective, constraint, function, goal).

**C** is a set of constraints which requires a proposal to be presented in order for an intent to be satisfied.

Definitions of objective, constraint, function, and goal are provided below.

1. **Objective** is a characteristic to be optimized by an artifact. It presents a measure against which the design is checked (e.g., minimize ecological impact by the bridge). Designers tend to use this class of intents as an evaluation which requires comparison among several competing designs.

   An objective is defined using the SHARED object model as a tuple of the form:

   **(Objective, oid, A, M, R, C)**

   where

A is a set composed of {(**Recommendation, agent**), (**Optimizer, action**), (**Recommendation, end**)}. **agent** is a recommendation which should achieve an end recommendation thorough an optimizing action. **agent** is an object attribute of the **Recommendation** class which refers to the recommendation achieving the condition sought by the designer. In the example, the **agent** is *the bridge*. **action** is an object attribute of the **Optimizer** class which refers to the kind of optimization sought by the designer. In the example, the **action** is *minimize*. **end** is an object attribute of the **Recommendation** class which refers to the optimized recommendation sought by the designer. In the example, the **end** is *the impact on the ecosystem.*

**M** is a set of methods which define the behavior of objectives. These methods provide objective's information retrieval and comparison, as well as mechanism for building the relationships between recommendations and objectives. Objectives require the comparison between several recommendations before they are considered satisfied and these methods allow for such implementation.

**R** is a set composed of {**Refers-to**}. **Refers-to** links a constraint with the intent that refers to it.

**C** is a set of constraints which require that at least two proposals be presented for the same objective in order to performed a comparison between them.

2. **Constraint** is a confinement or restriction on an artifact. It represents a boundary which the artifact or associated features should not surpass. For example, Article

8.9.3.1 of AASHTO-89 specifies a maximum deflection on a bridge to be 1/800th of the span. In this class of intents, designers only need to test if the criteria is met by the design recommendation without the need to compare it to other design alternatives.

A constraint is defined using the SHARED object model as a tuple of the form:

**(Constraint, oid, A, M, R, C)**

where

> **A** is a set composed of {(**Recommendation, agent**), (**Operator, comparison**), (**Recommendation, end**)}. **agent** is an object attribute of the **Recommendation** class which refers to the recommendation that should achieve the condition sought by the designer. In the example, the **agent** is *the bridge deflection.* **comparison** is an object attribute of the **Operator** class which refers to the kind of comparison between the **agent** and the **end**. In the example, the **comparison** is *maximum.* **end** is an object attribute of the **Recommendation** class which refers to the condition to be achieved. In the example, the **end** is *the 1/800th of the span.* The Article 8.9.3.1 of AASHTO-89 specification is recoded as a justification for the existence of the constraint.
>
> **M** is a set of methods which define the behavior of constraints. These methods provide constraint's information retrieval and comparison, as well as mechanism for building the relationships between recommendations and constraints. Constraints require a limit to be met and these methods allow for such implementation.

**R** is a set composed of {**Refers-to**}. **Refers-to** links a constraint with the intent that refers to it.

**C** is a set of constraints which tests the limits being defined. Note that this constraint component is part of the constraint intent.

3. **Function** is an action or activity performed by an artifact, e.g., to safely withstand the loads of the bridge during its lifetime. This class establishes the performance criteria that latter translates to the behavior of the system and specific constraints.

A function is defined using the SHARED object model as a tuple of the form:

**(Function, oid, A, M, R, C)**

where

**A** is a set composed of {(**Recommendation**, **agent**), (**Activity**, **action**), (**Set-Conditional**, **conditions**)}. **agent** is an object attribute of the **Recommendation** class which refers to the recommendation that should perform the action or activity. In the example, the **agent** is *the bridge*. **action** is an object attribute of the **Activity** class which refers to the activity to be performed by the **agent**. In the example, the **action** is *to withstand the load.* **conditions** is a set of object attributes of the **Conditional** class which refers to under which condition the **agent** should perform the **action**. In this example, the **condition** is *safe during the bridge lifetime.*

**M** is a set of methods which define the behavior of functions. These methods provide function's information retrieval and comparison, as well as mechanism for building the relationships between recommendations and functions. Functions require translation from action or activity to artifact behavior and these methods allow for such implementation.

**R** is a set composed of {**Refers-to**}. **Refers-to** links a constraint with the intent that refers to it.

**C** is a set of constraints requiring that sub-functions exist if a **system** is recommended. Since systems are functional abstraction of components, functions associated with systems should have component's functions associated with them.

4. **Goal** is a task to be achieved during the design. A goal has a plan of actions that leads to its achievement, and thus it can *consist-of* other goals (i.e., sub-goals). For example, a goal put forward by the structural engineer might be to select the material of a bridge. This class of intents establishes the process followed by designers.

A goal as modeled by DRIM augments the definition of CONGEN goal [Gorti et al., 1993], and it is defined as a tuple of the form:

**(Goal, oid, A, M, R, C)**

where

A is a set composed of {**CONGEN-Goal.A, A'**}. **CONGEN-Goal.A** refers to the name of the goal. **A'** is a set composed of {(**Designer, agent**), (**Activity, action**)}. **agent** is an object attribute of the **Designer** class which refers to the designer that is expected to perform the task. **action** is an object attribute of the **Activity** class which defines the task to be performed.

**M** is a set composed of {**CONGEN-Goal.M, M'**}. **CONGEN-Goal.M** includes methods on how the goal can be achieved. **M'** is a set of methods which define the behavior of goals. These methods provide goal's information retrieval and comparison, as well as mechanism for building the relationships between recommendations and goals. Goals require a task to be achieved and these methods allow for the check of the task performance.

**R** is a set composed of {**CONGEN-Goal.R, R'**}. **CONGEN-Goal.R** involves relationship to the parent plan. **R'** is a set composed of {**Refers-to, Part-of, Consists-of**}. **Refers-to** links a constraint with the intent that refers to it. **Part-of** links a goal with its parent plan. **Consists-of** links two goals in which one is the parent goal and the other is a sub-goal.

**C** is a set of constraints which involve the relationship with other goals and the parent plan.

### 5.2.4 Recommendation

Recommendation refers to the entity that satisfies the design intents. This construct can introduce or modify a design intent, a plan, or an artifact. When a recommendation introduces a design intent, it defines other things that need to be satisfied in order to achieve the design intent. When a plan is recommended, the set of goals to be achieved are introduced. When an artifact is recommended, SHARED constructs are used [Wong and Sriram, 1993b]. In the SHARED model, there are systems (e.g., bridge system) which are composed of components (e.g., the slab of the bridge). Components are functional abstractions of physical objects. Physical objects in turn represent the set of points that form the geometrical representation of an artifact (e.g, the set of coordinates that represent the volume of the slab). When a recommendation modifies an existing design intent, plan or artifact, their attributes or their values are changed.

A recommendation is defined using the SHARED object model as a tuple of the form:

**(Recommendation, oid, A, M, R, C)**

where

**A** is a set composed of {(**Proposal, proposal**)}. **proposal** is an object attribute of the **Proposal** class. **proposal** refers to the proposal which is responsible for the recommendation.

**M** is a set of methods which define the behavior of recommendations. These methods provide recommendation's information retrieval and comparison.

**R** is a set composed of {**Is-referred-by, Introduces, Modifies** }. **Is-referred-by** links a recommendation with the context in which it is defined. **Introduces** links a recommendation with the intent, the plan, or the artifact that it is introducing. **Modifies** links a recommendation with the intent, the plan, or the artifact that it is modifying.

**C** is a set of constraints which restricts recommendations to plans, artifacts, or intents.

A plan as modeled by DRIM augments the definition of CONGEN plan [Gorti et al., 1993], and is defined as a tuple of the form:

**(Plan, oid, A, M, R, C)**

where

**A** is a set composed of {**CONGEN-Plan.A**}. **CONGEN-Plan.A** is the set composed of {(**String, name**)} which is a character string representation of the plan's name.

**M** is a set composed of {**CONGEN-Plan.M, M'** }. **CONGEN-Plan.M** includes methods for scheduling and ordering goals to be achieved. CONGEN allows rules to achieve the planning based on the current design context conditions. **M'** is a set of methods which define the behavior of plans. These methods provide plan's information retrieval and comparison.

**R** is a set composed of {**Consists-of, Introduces, Modifies** }. **Consists-of** links a plan with the goals that are part of the plan. **Introduces** links a plan with the recommendation that introduces it. **Modifies** links a plan with the recommendation that modifies it.

**C** is a set composed of {**CONGEN-Plan.C**}. **CONGEN-Plan.C** is a set of constraints which enforce the order of goals.

An artifact as modeled by DRIM uses some elements of the definition of CONGEN artifact [Gorti et al., 1993] and is defined as a tuple of the form:

**(Artifact, oid, A, M, R, C)**

where

A is a set composed of {**CONGEN-Artifact.P**}. **CONGEN-Artifact.P** is the set of attributes that refer to the knowledge domain plans for designing the artifact.

**M** is a set of methods which define the behavior of artifacts. These methods provide artifact's information retrieval and comparison.

**R** is a set composed of {**Introduces, Modifies, Part-Of**}. **Introduces** links an artifact with the recommendation that introduces it. **Modifies** links an artifact with the recommendation that modifies it. **Part-Of** links an artifact with its sub-systems and components.

**C** is a set of constraints which limits an artifact to be introduced through a recommendation only.

A system as modeled by DRIM is a sub-class of the SHARED system [Wong and Sriram, 1993a] and the DRIM artifact. This multiple inheritance augments SHARED system abstraction with design rationale relationships. DRIM system is defined as a tuple of the form:

**(System, oid, A, M, R, C)**

where

**A** is a set composed of {**SHARED-System.A, Artifact.A**}. **SHARED-System.A** is the set of attributes related to the structure of the system. **Artifact.A** is the set of all the attributes from the **Artifact** class.

**M** is a set composed of {**SHARED-System.M, Artifact.M, M'**}. **SHARED-System.M** includes methods for defining spatial queries and transformations.

**Artifact.M** is the set of all the methods from the **Artifact** class. **M'** is a set of methods related to the systems' information retrieval and comparison.

**R** is a set composed of {**SHARED-System.R, Artifact.R, R'**}. **SHARED-System.R** is a set of spatial and composition relationships. **Artifact.R** is the set of all the relationships from the **Artifact** class. **R'** is a set composed of {**Consists-of** }. **Consists-of** links a system with other artifacts that could be either another system or a component.

**C** is a set composed of {**SHARED-System.C, Artifact.C**}. **SHARED-System.C** involves consistency checking. **Artifact.C** is the set of all the constraints from the **Artifact** class.

A component as modeled by DRIM is a sub-class of SHARED component [Wong and Sriram, 1993a] and the DRIM artifact. This multiple inheritance augments the SHARED component abstraction with design rationale relationships. DRIM component is defined as a tuple of the form:

**(Component, oid, A, M, R, C)**

where

**A** is a set composed of {**SHARED-Component.A, Artifact.A**}. **SHARED-Component.A** is the set of attributes related to the structure of the component. **Artifact.A** is the set of all the attributes from the **Artifact** class.

**M** is a set composed of {**SHARED-Component.M, Artifact.M, M'**}. **SHARED-Component.M** includes methods for defining spatial queries and transformations. **Artifact.M** is the set of all the methods from the **Artifact**

class. **M'** is a set of methods related to the components' information retrieval and comparison.

**R** is a set composed of {**SHARED-Component.R**, **Artifact.R**}. **SHARED-Component.R** is a set of spatial and composition relationships. **Artifact.R** is the set of all the relationships from the **Artifact** class.

C is a set composed of {**SHARED-Component.C**, **Artifact.C**}. **SHARED-Component.C** involves consistency checking. **Artifact.C** is the set of all the constraints from the **Artifact** class.

A physical-object as modeled by DRIM maps directly to the definition of SHARED physical-object; the reader is referred to [Wong and Sriram, 1993a] for a detailed description of various SHARED primitives.

### 5.2.5 Justification

Justification refers to a reason that partially explains why a recommendation will satisfy a design intent. A justification could be a *rule* (e.g., if a span is between 200 and 250 feet a two span bridge is feasible), a *case* (e.g., this bridge is similar to the Inn River Bridge in Switzerland), a *catalog* (e.g., this bridge was taken from an entry of "Standard Plans for Highway Bridges"), a *principle* (e.g., this relationship between measuring shear stress and fluid velocity is supported by the Law of the Wall), an *authority* (e.g., this shape has been suggested by someone who is an authority in the field), a *trade-off* (e.g., this is the best design based on the trade off between minimizing the cost and minimizing deflection), a *prototype* (e.g., the prototype of the ceramic valve produced these measurements), a *constraint network* (e.g., this condition satisfies all the constraints imposed on the system), or a *pareto optimal surface* (e.g., this design falls on the surface of best possible design when optimizing cost, schedule, and ecological impact). A justification may support other justifications by presenting supporting evidence or assumptions. This classification is derived from the case studies performed during this research. Interviewed designers prompted instances of those classes as reasons for presenting a recommendation. The designers also implied that there are relevant differences in their structure and use.

A justification and its sub-classes are defined using the SHARED object model as a tuple of the form:

**(Justification, oid, A, M, R, C)**

where

**A** is a set composed of {(**Proposal, proposal**)}. **proposal** is an object attribute of the **Justification** class. **proposal** refers to the proposal that introduces this justification.

**M** is a set of methods which define the behavior of justifications. These methods provide justifications' information retrieval and comparison.

**R** is a set composed of {**Is-related-to, Is-part-of**}. **Is-related-to** links a justification to the context in which it is referred. **Is-part-of** links a justification with the proposal that it is supporting.

**C** is a set of constraints which restricts the justification to be part of a proposal.

A rule is defined using the SHARED object model as a tuple of the form:

**(Rule, oid, A, M, Justification.R, Justification.C)**

where

A is a set composed of {**Justification.A**, **A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {(**Set-if**, **if**),(**Set-then**, **then**) }. **if** is a set of object, attribute as well as and/or operators that are pre-conditions to the rule. **then** is a set of object, attribute and operators that are post-conditions to the rule.

M is a set composed of {**Justification.M**, **M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** is a set of methods which define the behavior of rules. These methods provide rules' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A case is defined using the SHARED object model as a tuple of the form:

**(Case, oid, A, M, Justification.R, Justification.C)**

where

A is a set composed of {**Justification.A**, **A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {(**Designer**, **author**), (**String**, **period**), (**String**, **place**), (**Set-Recommendation**, **proposed-recommendations**), (**Set-Conditional**, **conditions**)}. **author** is an object attribute of the **Designer** class who performed the design. **period** is a character string representation of the time when the design took place. **place** is a character string representation of where the design took place. **proposed-recommendations** is a set of object attribute of the **Recommendation** class which represent the recommendation given in that design. **conditions** is a set of object attributes of the **Conditional** class which refers to the conditions under which the **agent** should perform the **action.** The attributes of the class **Case** are selected such that the rationale for the **proposed-recommendations** is captured. In the event that the case was provided through an user interface, then attributes such as **author**, **period**, **place**, and **conditions** give a context in which the recommendations was presented. However, in the event that the case is from previous design within the DRIM model, then those attributes are empty but the rationale for the recommendations is provided by the rationale links of the DRIM model itself.

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** is a set of methods which define the behavior of cases. These methods provide cases' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A catalog is defined using the SHARED object model as a tuple of the form:

**(Catalog, oid, A, M, Justification.R, Justification.C)**

where

A is a set composed of {**Justification.A, A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {(**String, document**), (**String, period**), (**String, place**), (**Set-Recommendation, proposed-recommendations**)}. **document** is a character string representation of where the design is documented. **period** is a character string representation of the time when the design took place. **place** is a character string representation of where the design took place. **proposed-recommendations** is a set of object attributes of the **Recommendation** class which represents the recommendation given in that design.

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M** is a set of methods which define the behavior of catalogs. These methods provide catalogs' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A principle is defined using the SHARED object model as a tuple of the form:

**(Principle, oid, A, M, Justification.R, Justification.C)**

where

A is a set composed of {**Justification.A, A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {(**String, document**), (**String, period**), (**String, place**), (**Set-formulae, formulae**), (**Set-Conditional, conditions**)}. **document** is a character string representation of where the design is documented. **period** is a character string representation of the time when the design took place. **place** is a character string representation of where the design took place. **formulae** is a set of object attribute of the **Formulae** class which represents the principles that justify the design. **conditions** is a set of object attributes of the **Conditional** class which refers to under which condition the **agent** should perform the **action.**

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M** is a set of methods which define the behavior of principles. These methods provide principles' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

An authority is defined using the SHARED object model as a tuple of the form:

**(Authority, oid, A, M, Justification.R, Justification.C)**

where

**A** is a set composed of {**Justification.A, A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {**(Designer, author), (String, period), (String, place), (String, title)**}. **author** is an object attribute of the **Designer** class who presents the recommendation. **period** is a character string representation of the time when the design took place. **place** is a character string representation of where the design took place. **title** is a character string representation of the title or status position of the **author.**

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** is a set of methods which define the behavior of authorities. These methods provide authorities' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A prototype is defined using the SHARED object model as a tuple of the form:

(**Prototype, oid, A, M, Justification.R, Justification.C**)

where

**A** is a set composed of {**Justification.A, A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {(**Designer, author**), (**String, period**), (**String, place**), (**String, scale**), (**Set-Recommendation, proposed-recommendations**), (**Set-Conditional, conditions**)}. **author** is an object attribute of the **Designer** class who performed the design. **period** is a character string representation of the time when the design took place. **place** is a character string representation of where the design took place. **scale** is a character string representation of the scale used in the prototype. **proposed-recommendations** is a set of object attributes of the **Recommendation** class which represent the recommendations given in that design. **conditions** is a set of object attributes of the **Conditional** class which refers to the conditions under which the **agent** should perform the **action**. The attributes of the class **Prototype** are selected such that the rationale for the

**proposed-recommendations** is captured. In the event that the prototype has been developed by other designers that did not use DRIM for the prototype development, then attributes such as **author**, **period**, **place**, **scale**, and **conditions** give a context in which the prototype was developed. However, in the event that the prototype is designed using DRIM constructs, then those attributes are empty but the rationale for the prototype is provided by the rationale links of the DRIM model itself.

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** is a set of methods which define the behavior of prototypes. These methods provide prototypes' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A constraint-network is defined using the SHARED object model as a tuple of the form:

(**Constraint-Network, oid, A, M, Justification.R, Justification.C**)

where

**A** is a set composed of {**Justification.A, (List-Constraint, constraints)**}. **Justification.A** is the set of all the attributes from the **Justification** class. **constraints** is a list of object attribute of the **Constraint** class in the order in which they must be satisfied.

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** includes methods for scheduling and ordering constraints to be achieved as well as the constraint-networks' information retrieval and comparison. This set of methods defines the behavior of constraint-networks.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

A pareto-optimal-surface is defined using the SHARED object model as a tuple of the form:

**(Pareto-Optimal-Surface, oid,** A, M, **Justification.R, Justification.C)**

where

A is a set composed of {**Justification.A, A'**}. **Justification.A** is the set of all the attributes from the **Justification** class. **A'** is a set composed of {**(Set-Recommendation, conditions), (UtilityMatrix, utilities)**}. **conditions** is a set of object attribute of the **Recommendation** class which refers to the recommendation proposed for the different conditions. **utilities** is an object attribute of the **UtilityMatrix** class which represent the interdependencies of the conditions.

**M** is a set composed of {**Justification.M, M'**}. **Justification.M** is the set of all the methods from the **Justification** class. **M'** is a set of methods which define the behavior of Pareto-optimal-surfaces. These methods provide Pareto-optimal-surfaces' information retrieval and comparison.

**Justification.R** is the set of all the relationships from the **Justification** class.

**Justification.C** is the set of all the constraints from the **Justification** class.

### 5.2.6 Context

A context represents the information generated during the design process. It represents all the recommendations made during the design. These recommendation can refer to an intent or an artifact or any of their attributes. The context is classified into two types: evidence and assumption. The evidence type refers to recommendations that are believed to be facts. For example, the geotechnical engineer may support his recommendation based on soil explorations which shows that there is clay at 100 feet depth. The assumption type refers to the elements or conditions that are assumed by the designer. For example, the construction manager may have assumed steel weight to be 6.5 psf for calculating transportation costs. When the structural engineer selects steel that weighs 10 psf, the contractor's assumption is violated and the transportation costs may change. Assumptions are data that have a certain degree of uncertainty attached to them. This uncertainty can be related to several sources: (1) the design agents; (2) the data quality; and (3) the design phase in which the data are created.

A context is defined using the SHARED object model as a tuple of the form:

**(Context, oid, A, M, R, C)**

where

**A** is a set composed of {(**String, uncertainty**)}. **uncertainty** is a character string representation of the reliability of the assertion.

**M** is a set of methods which define the behavior of contexts. These methods provide contexts' information retrieval and comparison.

**R** is a set composed of {**Is-related-to, Is-referred-by**}. **Is-related-to** links a context to the **justification** to which relates. **Is-referred-by** links a context with the **Recommendation** to which refers.

**C** is a set of constraints which involve access to the latest version of an object.

An evidence is defined using the SHARED object model as a tuple of the form:

**(Evidence, oid, A, M, Context.R, Context.C)**

where

**A** is a set composed of {(**String, uncertainty, "none"**)}. Evidence sets the value of the attribute **uncertainty** of the class **Context** to "none" since the assertion is based on factual information.

**M** is a set composed of {**Context.M, M'**}. **Context.M** is the set of all the methods from the **Context** class. **M'** is a set of methods which define the behavior of evidences. These methods provide evidences' information retrieval and comparison.

**Context.R** is the set of all the relationships from the **Context** class.

**Context.C** is the set of all the constraints from the **Context** class.

An assumption is defined using the SHARED object model as a tuple of the form:

**(Assumption, oid, A, M, Context.R, Context.C)**

**A** is a set composed of {(**String, uncertainty**)}. Assumption does not set the value of the attribute **uncertainty** of the class **Context**. This value is obtained from the degree of reliability of the information (from 0 to 100) as set by the designer.

**M** is a set composed of {**Context.M**, **M'**}. **Context.M** is the set of all the methods from the **Context** class. **M'** is a set of methods which define the behavior of assumptions. These methods provide assumptions' information retrieval, comparison, and validity check. The methods related to the validity check of the assumptions maintain a record of when the assumptions were generated and of when they were used. This record allows the methods to compare assumption information with the factual data when that become available. In this way, the system can flag discrepancies between the assumptions and the real data.

**Context.R** is the set of all the relationships from the **Context** class.

**Context.C** is the set of all the constraints from the **Context** class.

## 5.3 DRIM Relationships

The primitives presented in the last section are not independent units. These primitives have a set of relationships defined between them. A detailed description of the relationships of the model is given as follows. Section 5.3.1 presents the *versions-of* relationship. Section 5.3.2 outlines the *is-alternative-of* relationship. The *consists-of* relationship is presented in Section 5.3.3. The *presents/based-on* relationship is explained in Section 5.3.4. Section 5.3.5 outlines the *refers-to* relationship. Sections 5.3.6 and 5.3.7 presents the *introduces* and the *modifies* relationships respectively. The *is-referred-to* and the *is-related-to* relationships are explained in Sections 5.3.8 and 5.3.9. Section 5.3.10

outlines the *reacts-to* relationship. Finally, the *negotiates-with* relationship is presented in Section 5.3.11.

### 5.3.1 Versions-of

The *version-of* relationship relates a proposal to another proposal which has a more detailed or developed recommendation than the first proposal. The new recommendation modifies an existing recommendation changing some of its attributes. *versions-of* relationships serve to keep track of the evolution of recommendations by recording the changes that are made to them. Thus, the *versions-of* relationship is important for design, since design is an experimental and incremental process. Recommendation changes performed at one step in the design are due to the results obtained from the previous recommendation.

Versions occur as a result of the introduction of new intents that need to be satisfied. These intents may introduce a different stage of the design process which does not necessarily imply change but increases detail on the recommendation. DRIM only allows modifications to be done through introduction of proposals. This important condition allows DRIM to keep track of the design rationale evolution. *Versions-of* relationships at the proposal level maintain the rationale trace behind a change, so that it can be retrieved later. DRIM uses the SHARED version relationship [Wong and Sriram, 1993a] for artifact evolution and underlying OODBM's version management facility for other types of objects.

An important characteristic of the *versions-of* link is that it allows the re-use of information. The *versions-of* relationship maps to the new version all the old proposal's unchanged attributes and their values. In addition, the changed attributes and values are incorporated into the new proposal together with a link to the old proposal. This behavior keeps a proposal complete at the current stage of the design while maintaining a trace to the old structure of the proposal.

A versions-of relationship is defined using the SHARED object model as a tuple of the form:

**(Versions_of, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Proposal, existing**), (**Proposal, version**)}. **existing** is an object of the **Proposal** class which refers to the old proposal being modified. **version** is an object of the **Proposal** class which refers to the new proposal replacing the old one.

**A** is composed of {(**Boolean, dependent**)}. **dependent** is used to control propagation of deletion of a versioned object. (i.e., whether the versioned object is deleted if **version** is deleted).

**M** is a set of methods which define the behavior of relationships. These methods provide relationships' information retrieval and comparison.

**C** is a set of constraints which check that the recommendations presented by the proposals satisfy the intent.

### 5.3.2 Is-Alternative-of

The *is-alternative-of* relationship is used to associate a proposal with another "similar" proposal which it can replace. Alternative proposals contain recommendations that are usually instances of different recommendation classes which satisfy the same intent. These recommendations represent different **technologies** or **processes** which can be used to satisfy some functional requirements of a design. Here, a technology is defined as a specific type of system together with its components, or a specific type of component. For example, a truss frame and a rigid frame are alternative types of frame that provide support for lateral loads. A process is defined as a specific plan together with its component goals or a specific intent that must be satisfied.

An important characteristic of the *is-alternative-of* link is that avoids the re-input of known information. The *is-alternative-of* relationship maps to the new proposal the links of the proposal that is being replaced. For example, a new alternative proposal satisfies the same intents as the old proposal and perhaps new intents. The *is-alternative-of* relationship maintains the established links and adds the new intents to the list of intents that the new proposal needs to satisfy. This behavior keeps a proposal complete at the current stage of the design while avoiding the repetition of work.

An is-alternative-of relationship is defined using the SHARED object model as a tuple of the form:

**(Is_alternative_of, Ro, A, M, C)**

where

**Ro** is a set composed of{({**Proposal**}, **alternative**), (**Intent, inte**)}. **alternative** is an object of the **Proposal** class which refers to the alternative proposal being presented. **inte** is an object of the **Intent** class which refers to the intent that is sought.

**A** is composed of {(**Boolean, dependent**), (**Recommendation, default_alter**)}. **dependent** is used to control propagation of deletion of a versioned object.

(i.e., whether the versioned object is deleted if **obj** is deleted). **default_alter** represents the default alternative recommendation object.

**M** is a set of methods which define the behavior of *is-alternative-of* relationships. These methods provide *is-alternative-of* relationships' information retrieval and comparison.

**C** is a set of constraints which checks whether the recommendations presented by the proposals satisfy the intent.

### 5.3.3 Consists-Of

The *consists-of* relationship defines a special relationship between a proposal and its sub-proposals. The sub-proposals when put together satisfy the intent sought by the parent proposal. That is, the *consists-of* relationship implements the notion of sub-dividing the problem. This concept of sub-dividing the problem is enforced by the relationship behavior to inform the designer when a sub-proposal has not been satisfied or it has been deleted.

A consists-of relationship is defined using the SHARED object model as a tuple of the form:

**(Consists_of, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Proposal, parent**), (**Proposal, sub**)}. **parent** is an object of the **Proposal** class which refers to the parent proposal. **sub** is an object of the **Proposal** class which refers to the new proposal that achieves in part the intent sought by the parent proposal.

A is composed of {(**Boolean, dependent**)}. **dependent** is used to control deletion of sub-proposals.

**M** is a set of methods which define the behavior of *consists-of* relationships. These methods provide *consists-of* relationships' information retrieval and comparison.

C is a set composed of constraints which involve the deletion of sub-proposals.

### 5.3.4 Presents/Based-on

The *presents/based-on* relationship relates a designer, an intent, and a proposal being presented. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. A *presents/based-on* relationship is defined using the SHARED object model as a tuple of the form:

**(Presents/Based-on, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Designer, presenter**), (**Intent, based-on**), (**Proposal, proposal**)}. **presenter** is an object of the **Designer** class which refers to the designer that is presenting the new proposal. **based-on** is an object of the **Intent** class which refers to the intent being sought. **proposal** is an object of the **Proposal** class which refers to the proposal presented.

A is composed of {(**Boolean, dependent**)}. **dependent** is used to control if a proposal may exist even when an intent is deleted.

**M** is a set of methods which define the behavior of *presents/based-on* relationships. These methods provide *presents/based-on* relationships' information retrieval and comparison.

**C** is a set composed of constraints which warns that a proposal should be revised if an intent changes. These constraints also deletes a proposal if the intent is deleted.

### 5.3.5 Refers-to

The *refers-to* relationship relates an intent with the type of need sought. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. A *refers-to* relationship is defined using the SHARED object model as a tuple of the form:

**(Refers-to, Ro, A, M, C)**

where

**Ro** is a set composed of {**(Intent, intent)**, **(Need, need)**}. **intent** is an object of the **Intent** class which refers to the intent being sought. **need** is an object of the **Need** class which is a super-class of the objective, constraint, function, and goal classes.

**A** is composed of {**(Boolean, dependent)**}. **dependent** is used to delete the **intent** when the **need** is removed.

**M** is a set of methods which define the behavior of *refers-to* relationships. These methods provide *refers-to* relationships' information retrieval and comparison.

C is a set composed of constraints which involve deletion of **need** when **intent** is removed.

### 5.3.6 Introduces

The *introduces* relationship relates a recommendation with item presented type. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. An *introduces* relationship is defined using the SHARED object model as a tuple of the form:

**(Introduces, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Recommendation, recommendation**), (**Recom, item**)}. **recommendation** is an object of the **Recommendation** class which refers to the recommendation given as a solution. **item** is an object of the **Recom** class which is a super-class of intent, plan, and artifact.

**A** is composed of {(**Boolean, dependent**)}. **dependent** is used to control the deletion of **recommendation** when **item** is removed.

**M** is a set of methods which define the behavior of *introduces* relationships. These methods provide *introduces* relationships' information retrieval and comparison.

**C** is a set composed of constraints which deletes the **item** when the **recommendation** is removed.

### 5.3.7 Modifies

The *modifies* relationship relates a recommendation with the item modified type. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. A *modifies* relationship is defined using the SHARED object model as a tuple of the form:

**(Modifies, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Recommendation, recommendation**), (**Recom, item**)}. **recommendation** is an object of the **Recommendation** class which refers to the recommendation given as a solution. **item** is an object of the **Recom** class which refers to either an intent, a plan, or an artifact that is modified.

**A** is composed of {(**Boolean, dependent**)}. **dependent** is used to delete the **recommendation** when the **item** is removed.

**M** is a set of methods which define the behavior of *modifies* relationships. These methods provide *modifies* relationships' information retrieval and comparison.

**C** is a set composed of constraints which involve deletion of **item** when **recommendation** is removed.

### 5.3.8 Is-referred-to

The *is-referred-to* relationship relates a recommendation with the context in which it is given. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. An *is-referred-to* relationship is defined using the SHARED object model as a tuple of the form:

**(Is-referred-to, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Recommendation, recommendation**), (**Context, context**)}. **recommendation** is an object of the **Recommendation** class which refers to the recommendation given as a solution. **context** is an object of the **Context** class. **context** refers to the context in which the recommendation is given.

**A** is composed of {(**Boolean, dependent**)}. **dependent** is used to delete the **recommendation** when the **context** is removed.

**M** is a set of methods which define the behavior of *is-referred-to* relationships. These methods provide *is-referred-to* relationships' information retrieval and comparison.

**C** is a set composed of constraints which deletes the **context** when the **recommendation** is removed.

### 5.3.9 Is-related-to

The *is-related-to* relationship relates a justification with the context in which it is given. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. An *is-related-to* relationship is defined using the SHARED object model as a

**(Is-related-to, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Justification, justification**), (**Context, context**)}. **justification** is an object of the **Justification** class which refers to the recommendation given as a solution. **context** is an object of the **Context** class which refers to the context in which the justification is given.

**A** is composed of {(**Boolean, dependent**)}. **dependent** is used to control the deletion of **justification** when **item** is removed.

**M** is a set of methods which define the behavior of *is-related-to* relationships. These methods provide *is-related-to* relationships' information retrieval and comparison.

**C** is a set composed of constraints which involve the deletion of **context** when **justification** is removed.

### 5.3.10 Reacts-to

The *reacts-to* relationship relates two proposals. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the designer of the inconsistency. A *reacts-to* relationship is defined using the SHARED object model as a tuple of the form:

**(Reacts-to, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Proposal, existing**), (**Proposal, opposite**)}. **existing** is an object of the **Proposal** class which refers to the old proposal. **opposite** is an object of the **Proposal** class which refers to a new proposal that reacts to the old one.

**A** is composed of {(**String, position**)}. **position** refers to the type of reaction between the proposals. **position** can be either support, contradict or change.

**M** is a set of methods which define the behavior of *reacts-to* relationships. These methods provide *reacts-to* relationships' information retrieval and comparison.

**C** is a set composed of constraints which involve creating a **negotiates-with** link with designers presenting the conflicting proposal. The **negotiates-with** link is removed by the computer when conflict is resolved.

### 5.3.11 Negotiates-with

The *negotiates-with* relationship relates designers who present conflicting proposals. This relationship is derived from the *reacts-to* relationship between proposals when the *position* is *contradicting*. When the position is of *support*, the *negotiates-with* relationship is not created. This relationship is only maintained when all the objects in the relationship exist in the current context. When any of them is deleted, the relationship immediately informs the remaining designer of the inconsistency.

A *negotiates-with* relationship is defined using the SHARED object model as a tuple of the form:

**(Negotiates-with, Ro, A, M, C)**

where

**Ro** is a set composed of {(**Designer**, **presenter**), (**Designer**, **opposed**)}. **presenter** is an object of the **Designer** class which refers to the designer that is presenting the new proposal. **opposed** is an object of the **Designer** class which refers to the designer who has the opposing view.

**A** is composed of {(**Integer**, **interactions**), (**Integer**, **agree**)}. **interactions** is the number of times that the two designers have interacted in the past. **agree** is the number of times the two designers have agreed.

**M** is a set of methods which define the behavior of *negotiates-with* relationships. These methods provide *negotiates-with* relationships' information retrieval and comparison.

**C** is a set composed of constraints which checks that **presenter** and **opposed** are different designers. If they are same, the relationship is not created.

## 5.4 Design with DRIM Objects

Previous sections described various DRIM primitive classes and relationships between these classes. This section presents how these classes and relationships are used to describe the design process. Thus, design can be defined as a set of operators that instantiate or modify objects of the different classes and relationships defined in DRIM according to the conditions in the process.

Based on these premises, design can be expressed at the most general level as the function[2]:

$$D(I,C) \Rightarrow P \tag{5.1}$$

where

- $D$ is the set of design operators or knowledge chunks that could be used during any particular design. In other words, $D$ represents the knowledge existing in a given domain that could be used to come up with a satisfactory design.
- $I$ is the set of intents that need to be satisfied by the designer or the product during the design. In other words, $I$ represents all the objectives, constraints, goals and functions that are to be achieved.
- $C$ is the context in which the design is performed. $C$ can be visualized as the environment in which the design is performed. Previous decisions bring solutions to the problem but may also bring new issues to be resolved. Thus, $C$ represents all the decisions that have been made during the design up to a particular point.
- $P$ is the set of proposals that are obtained using design operators or knowledge $D$ over the intents $I$ needed to be satisfied taking in consideration the context $C$ of the design.

[2] Symbols used in the following formulae are explained in the table of symbols at the beginning of the thesis.

Equation 5.1 is a general view of design. However, design is performed at different levels of detail using an opportunistic approach. This is reflected in the following equation:

$$\forall i_\ell \in I_\ell, \exists d \subset D : \forall d_\ell \in d, d_\ell(i_\ell, C) \Rightarrow [p_{\ell_m} | add\_justification(p_{\ell_m})] \tag{5.2}$$

where

$$\begin{aligned} p_{\ell_m} &\in P_\ell, & (5.3)\\ p_{\ell_m} &= \{r_{\ell_m}, d_{\ell_m}\}, & (5.4)\\ d_{\ell_m} &\equiv \{\cup d_\ell : d_\ell(i_\ell, C) \Rightarrow [p_{\ell_m} | add\_justification(p_{\ell_m})]\}, & (5.5)\\ r_{\ell_m} &= \mathcal{O}(i_{\ell+1} | plan_{\ell+1} | a_{\ell+1}) & (5.6) \end{aligned}$$

- $i_\ell$ is an intent at level $\ell$ of the hierarchy.
- $I_\ell$ is the set of all the intents at level $\ell$ of the hierarchy.
- $d$ is a subset of operators from the $D$ set of operators available to the designer.
- $d_\ell$ is an operator in $d$ that can satisfy $i_\ell$ within the context $C$ and leads to a proposal.
- $p_{\ell_m}$ is a proposal that is in the set of all the proposals at level $\ell$.
- $add\_justification(p_{\ell_m}, d_\ell)$ is a function that adds $d_\ell$ as a justification to a proposal that already has been created.
- $P_\ell$ is the set of all the proposals presented at level $\ell$ of the hierarchy.
- $r_{\ell_m}$ represents the recommendation presented by the proposal.
- $d_{\ell_m}$ is a set of all $d_\ell$ that are justifications to a proposal $p_{\ell_m}$.

- $i_{\ell+1}$, $plan_{\ell+1}$, and $a_{\ell+1}$ are intent, plan, and artifact at level $\ell + 1$. This level may be higher or lower in the hierarchy depending on whether a *top-down* or *bottom-up* design approach is followed respectively.

Section 2.3 presents the different design stages. Each one of these stages differ in the recommendation presented by the proposal. The recommendation that a proposal presents is limited according to the stage in which the designer is operating. Section 2.3 also mentions that the process is iterative and that the stages do not follow a time line –on the contrary, they can be chaotic and intermixed. DRIM supports that iteration and permits identification of each stage at any given point by the type of recommendation presented through a proposal.

For example, the *feasibility analysis* stage is concerned with the development of the intents that need to be satisfied. Thus, $r_{\ell_m}$ is defined as follows:

$$r_{\ell_m} = introduces(i_{\ell+1}). \tag{5.7}$$

The *problem identification and formulation* phase is concerned with the scope of and specifications of the intents. Thus, $r_{\ell_m}$ is expressed as:

$$r_{\ell_m} = modifies(i_{\ell+1}). \tag{5.8}$$

The *preliminary design* phase focuses primarily on the generation of concepts to satisfy various requirements. Thus, $r_{\ell_m}$ is presented as:

$$r_{\ell_m} = introduces(a_{\ell+1}). \tag{5.9}$$

The *optimization* phase focuses primarily on refining the concepts presented on the *preliminary design* phase. Thus, $r_{\ell_m}$ is presented as:

$$r_{\ell_m} = modifies(a_{\ell+1}). \tag{5.10}$$

The *detailed design* phase is concerned with the laying out the plan for the construction of the product. Thus, $r_{\ell_m}$ is presented as:

$$r_{\ell_m} = introduces(plan_{\ell+1}). \tag{5.11}$$

The *construction* phase is concerned with the implementation of the plan and its refining. Thus, $r_{\ell_m}$ is presented as:

$$r_{\ell_m} = modifies(plan_{\ell+1}). \tag{5.12}$$

In any of these processes there are four activities that are performed. These activities are *identification*, *analysis*, *evaluation*, and *selection.* DRIM also allows for these activities to be represented through the use of goals and plans. They follow the patterns outlined below:

*Identification* is the task of finding the recommendation that satisfies a set of intents. This step is presented in [Gorti et al., 1993]. Gorti *et al.* hypothesize that the process by which designers identify possible satisfying solutions to their intents is to present the set of behaviors expected for an artifact that would meet such an intent. Then, after the expected behaviors have been defined, the designers search for artifacts that have such behaviors. This two step approach leads to the artifacts recommended. Gorti *et al.* approach's is limited to the artifact. DRIM supports not only the recommendation of an artifact but also the recommendation of intents. For this case, DRIM has to have a representation of the intent's effect. The intent's effect is analogous to the behavior on the artifact. Thus, the task of identification is represented as :

$$\forall i_\ell \in I_\ell, \mathcal{B}(i_\ell, C) \Rightarrow [r_{\ell_m}.[e|b] \;\&\; \mathcal{R}(r_{\ell_m}.(e|b), C)] \Rightarrow r_{\ell_m}.s \qquad (5.13)$$

where

- $\mathcal{B}$ is an operator that identifies the behavior or effect associated with an intent.
- $r_{\ell_m}.[e|b]$ is the behavior or effect of the recommendation at level $\ell$ produced by the intent. $e$ or $b$ depends on whether the recommendation is an intent or an artifact respectively.
- $\mathcal{R}$ is the operator that identifies the structure associated with the recommendation according to the expected behavior or effect.
- $r_{\ell_m}.[e|b]$ is the structure of the recommendation at level $\ell$ produced by the intent.

*Analysis* is the task of determining the behavior or effect of a recommendation. Once the structure of the recommendation has been selected the behavior or effect of the recommendation is found by applying a set of operators. Thus, the task of analysis is represented as:

$$\forall r_{\ell_m} \in R_\ell, \{\mathcal{A}(r_{\ell_m}.s, C) \Rightarrow r_{\ell_m}.[e|b]\} \qquad (5.14)$$

where

- $\mathcal{A}$ is an operator that produces the behavior or effect of a recommendation. This operator can be a mathematical equation or a computer program such as ABAQUS [Company, 1990].

*Evaluation* is the task of determining how well the behavior or effect of a recommendation matches the expected behavior or effects of an intent. Once the behavior or effect of the recommendation has been selected an operator is used to assess the degree of satisfaction according to the expected behavior or effect. Thus, the task of evaluation is represented as:

$$\forall r_{\ell_m} \in R_\ell, \exists i_\ell \in I_\ell : \mathcal{E}(r_{\ell_m}.[e|b], i_\ell, C) \Rightarrow i_\ell.ds \tag{5.15}$$

where

- $\mathcal{E}$ is an operator that determines the degree by which the behavior or effect of a recommendation satisfy an intent.
- $i_\ell.ds$ is the degree by which the intent related to the recommendation being evaluated is satisfied.

*Selection* is the task of selecting recommendations that satisfy an intent. Thus, the task of selection is represented as:

$$\forall r_{\ell_m} \in R_\ell. \exists i_\ell \in I_\ell : \mathcal{S}(\{r_{\ell_m}\}, \{i_\ell.ds, i_\ell.r\}, C) \Rightarrow \{r_{\ell_m}\} \tag{5.16}$$

where

- $\mathcal{S}$ is an operator that determines which of the recommendations is the best for satisfying a set of intents.
- $i_\ell.r$ is the rank of the intent.

## 5.5 Design Rationale Trace

The design rationale trace is the linkage graph of the different proposals presented during a recommendation generation. In the trace, each recommendation is only introduced by a proposal that supports its entry into the design domain. This proposal tends to be one of selection. However, selection is substantiated by proposals from the evaluation, analysis, and identification kind. These kinds of proposals can take different roles during the process. For example, evaluation proposals may contradict or support other proposals already presented. An important feature in the DRIM model is that proposals can only contradict other proposals based on support for the removal of the solution or support for an alternate solution.

Design rationale is defined as:

$$\forall r_{\ell_m} \in R_\ell, \exists p_\ell \subset P_\ell : Rationale(r_{\ell_m}) \Rightarrow Trace(p_{\ell_m}) \tag{5.17}$$

where

- *Trace* is the connection of all the proposals that support the presentation of $r_{\ell_m}$.
- *Rationale* is the link to all the supporting proposals.

The following examples illustrate how the *Trace* connection works.

$$\begin{aligned} t_0 \Rightarrow d_{ab}(i_0, C) \quad &\Rightarrow \quad P_0 \\ where \quad P_0 &= \{i_1, d_{ab}\} \end{aligned} \tag{5.18}$$

$$t_1 \Rightarrow d_{ac}(i_0, C) \quad \Rightarrow \quad P_1 \qquad (5.19)$$
$$where \quad P_1 = \{i_2, d_{ac}\}$$

- Equations 5.18 and 5.19 introduce intents $i_1$ and $i_2$ as sub-intents of intent $i_0$. These means that the solution to the design problem has to satisfy those two intents in order to achieve intent $i_0$. $t$ represents the time at which an assertion was made. $d$ represents the particular design knowledge that was used for presenting the proposal (i.e., the rule or the case that supports the proposal). $P$ represents the proposal presented. $C$ represents the context in which the proposal is made.

$$t_2 \Rightarrow d_{ba}(i_1, C) \quad \Rightarrow \quad P_2 \qquad (5.20)$$
$$where \quad P_2 = \{r_1, d_{ba}\}$$

- Equation 5.20 introduces a recommendation that satisfies $i_1$ based on the justification given by $d_{ba}$.

$$t_3 \Rightarrow d_{cd}(i_2, C) \quad \Rightarrow \quad (P_3 \,\&\, C'(P_3, P_2)) \qquad (5.21)$$
$$where \quad P_3 = \{\neg r_1, d_{cd}\}$$

- Equation 5.21 contradicts the proposal presented in Equation 5.20 by presenting a recommendation to remove $r_1$. This recommendation is based on the need to satisfy $i_2$ and the justification $d_{cd}$ is given as a reason of why $r_1$ does not satisfy $i_2$. & represents the and operator. $C'$ represents the reacts-to relationship with a contradicting position. $\neg$ represents the negation of the recommendation.

$$t_4 \Rightarrow d_{dc}(i_2, C) \quad \Rightarrow \quad (P_4 \,\&\, S(P_4, P_2)) \,\&\, C'(P_4, P_3) \qquad (5.22)$$

$$where \quad P_4 = \{r_1, d_{cd}\}$$

- Equation 5.22 supports the use of recommendation $r_1$ to satisfy $i_2$ based on the justification $d_{dc}$. Thus, it contradicts the proposal presented in Equation 5.21. $S$ represents the reacts-to relationship with a supporting position.

$$t_5 \Rightarrow d_{ef}(i_2, C) \quad \Rightarrow \quad (Add\_Justification(P_3^{'} \ \& \ (P_3^{'}, P_4))) \tag{5.23}$$
$$where \quad P_3^{'} = \{\neg r_1, \{d_{cd}, d_{ef}\}\}$$

- Equation 5.23 adds a new justification $d_{ef}$ to the proposal presented in Equation 5.21. Thus creating a new object version of the proposal with the new information. It also contradicts the proposal presented in Equation 5.22. $P'$ represents a proposal object that has been versioned.

$$t_6 \Rightarrow d_{fe}(i_2, C) \quad \Rightarrow \quad (P_5 \ \& \ S(P_5, P_3^{'}) \ \& \ V(P_5, P_3^{'})) \tag{5.24}$$
$$where \quad P_5 = \{r_1^{v_1}, d_{fe}\}$$

- Equation 5.24 presents a version of the recommendation as introduced by a proposal.[3] This proposal supports the proposal presented in Equation 5.23 and build a versions-of link to the proposal presented in equation Equation 5.22. $V$ represents the versions-of relationship between two proposals.

$$t_7 \Rightarrow d_{dg}(i_1, C) \quad \Rightarrow \quad (P_6 \ \& \ S(P_6, P_3^{'})) \tag{5.25}$$
$$where \quad P_5 = \{r_1^{v_1}, d_{dg}\}$$

[3] This version is the version of the recommendation meaning incremental change. The version referred to in the previous paragraph is the version of the object.

- Equation 5.25 presents a proposal with the same recommendation as the one presented in Equation 5.24 but trying to satisfy intent $i_1$.

$$\begin{aligned} t_8 \Rightarrow d_{vf}(i_0, C) \quad &\Rightarrow \quad (P_7 \ \& \ C(P_7, P_1)) \\ where \quad & P_7 = \{\neg i_2, d_{vf}\} \end{aligned} \tag{5.26}$$

- Equation 5.26 presents a proposal that recommends the removal of $i_2$. Thus, it contradicts the proposal presented in Equation 5.19.

$$\begin{aligned} t_9 \Rightarrow d_{ba}(i_1, C) \quad &\Rightarrow \quad (P_2' \ \& \ S(P_2', P_7)) \\ where \quad & P_2' = \{r_1, d_{ba}\} \end{aligned} \tag{5.27}$$

- Equation 5.27 presents a version of the proposal object presented in Equation 5.20 supporting the proposal presented in Equation 5.26.

$$\begin{aligned} t_x \Rightarrow d_{hj}(i_{34}, C) \quad &\Rightarrow \quad P_{28} \\ where \quad & P_{28} = \{r_{30}^{as}, d_{hj}\} \end{aligned} \tag{5.28}$$

- Equation 5.28 presents a proposal recommending a value that has not been set yet but it is assumed. $r^{as}$ represents a recommendation that has been assumed.

These sets of equations show how the DRIM model captures the rationale of a recommendation when there are versions, conflicts, and assumptions involved. The rationale depends on the stage of the design as denoted by the time variable $t$. For example, at time $t_2$ the rationale for having $r_1$ is because of the proposal in Equation 5.20. Then, the reason for having to satisfy intent $i_1$ may be asked. This question can be answered by the proposal presented in Equation 5.19.

At time $t_6$, a question may be arise concerning why $r_1$ has not been selected. The answer is that the proposal in Equation 5.22 that introduces it is in contradiction with the proposal in Equation 5.23, resulting in the improved version presented in Equation 5.24.

Finally, at time $t_9$, a designer may ask why in that case $r_1$ is now being used. The answer is that the proposal in Equation 5.20 that introduces it supports the proposal of removing $i_2$ which was an intent that $r_1$ does not fully satisfy.

## 5.6 Conclusions

This chapter presents a design rationale model that allows us to record explicitly the design process. The model captures all the steps which designers go through during the design of an artifact. These steps were described in terms of mathematical terms that exemplify the operators and relationships that are used during the design.

One advantage of the design rationale model discussed was that no part of the design process was left implicit. This representation allows us to judge the relevance and validity of each step in conjunction with the whole. Each proposal made by the designers must be supported with justifications (i.e. the assumptions and the conditions in which these proposals are valid). When context changes and re-evaluation of the proposals are necessary, previous proposals have to be re-linked under the condition that the past links are maintained with old versions of the objects. Thus, reuse of design knowledge is possible since only the parts that change are reviewed. However, this model also gives us the sense of security that any proposal that needs review will be reviewed. Since the relationships between proposals are explicitly stated, propagation of change is possible.

Another advantage of the model is that the causes of variance from what is expected from the design artifact can be presented. By navigating through the model objects, a computer system based on this model can find the areas or points where there is any contradiction. The cause of a problem is derived from the inconsistency of the proposals related to the problem. Next chapter explains in detail how SHARED-DRIM can use the model for capturing and supporting conflict mitigation.

# Chapter 6

# SHARED-DRIMS Conflict Mitigation System

*Conflicts . . . arise because of differing objectives . . . among different interest groups or institutions. Frequently each party in conflict fails to comprehend fully the interdependence of the complex issues and institutional objectives and constraints in conflict. So it becomes a learning process . . .*

Daniel P. Loucks , *[Loucks, 1989]*

## 6.1 Introduction

To demonstrate the feasibility and utility of the model for representing the design rationale, a system is needed which can capture and use such information. Project information management systems[1] typically have been passive. They provide support for storing and

[1] Project information management systems are called management systems throughout the chapter.

modifying large amounts of project data, but they lack support for generating, analyzing and understanding this data. The responsibility of understanding, generating and analyzing the data is left to the human user.

The amount of data to be analyzed by the user is copious. The user may spend an enormous amount of time analyzing data that may not be important. The user may know that a conflict exists, but have no easy way to determine its causes. Without knowing the cause, the user cannot suggest an appropriate course of action. Hence, management systems must take a more active role in supporting the human decision-making activity. Management systems need to understand the data stored and provide support for presenting design alternatives, as well as to present the rationale for the decisions made by the designer. Management systems also need to be able to detect inconsistencies in the data, as well as report the cause of this inconsistency. A project management system must be able to answer questions about the stored data such as:

- Why is the material steel?
- Why is the project one month late?
- Why is the foundation cost greater than the estimated cost?
- Who is responsible for cost overruns?

This chapter proposes SHARED-DRIMS, a proactive project management system that not only will be able to answer questions about the stored data, but will also be able to react to data inconsistency and then help designers to mitigate conflicts presented by the inconsistencies. In order to meet the needs of such a proactive system, the current representation of the knowledge used in management systems must be modified.

Knowledge about the design rationale should be explicitly represented to permit the test of inconsistencies in a different context. The design rationale model presented in Chapter 5 provides the representational tools for such a proactive project management system.

The following sections describe the functionalities of SHARED-DRIMS. Section 6.2 outlines the architecture of SHARED-DRIMS. This section also presents the system's base and design rationale modules. Section 6.3 describes the strategies used by the system in capturing design rationale. These strategies include retrieval from design support systems, from previous designs, or from user's inputs. Section 6.4 outlines the functions used by the system for conflict mitigation. These functions cover conflict detection, causes, resolution and prevention. Finally, Section 6.5 summarizes the salient points discussed in this chapter.

## 6.2 SHARED-DRIMS Architecture

Figure 6-1 presents the general organization of SHARED-DRIMS. This system consists of satellite modules attached to each design agent [Sriram et al., 1989], and a central core connected to all the design agents involved. Each design agent participating in the process has: 1) a knowledge base, 2) a product model, and 3) a design intent-recommendation model (DRIM). The knowledge base consists of all the relevant information (e.g., rules, cases, etc.) that influence any recommendation generated in the professional domain. The product model contains a representation of the artifact at different levels of abstraction in that domain. DRIM consists of three primary components: 1) the desired intents; 2) the recommendations believed to satisfy the intents; and 3) the justifications used by the designer in order to consider those recommendations. In the central core of the model, we have: 1) a physical representation of the product; 2) a knowledge base; and 3) a negotiated DRIM. The physical product representation presents the geometric description of the artifact and its parts. The knowledge base consists of the relevant information related to the interactions between different design agents in the design process. The negotiated DRIM records all intents and reasons generated during a negotiation process.

SHARED-DRIMS helps in negotiation and conflict mitigation by providing access to individual designers' design rationale and information about past negotiations. The following section describes various SHARED-DRIMS modules.

### 6.2.1 Base Modules

SHARED-DRIMS is part of a framework which provides the components of a total design environment. The different modules used in conjunction with SHARED-DRIMS are independent systems developed as part of the DICE project [Sriram et al., 1989]. These modules are developed using object-oriented methodology and supported by an object-oriented database management system (OODBMS). The OODBMS allows the objects to be functionally augmented. SHARED-DRIMS creates wrappers around the objects from the basic modules. It also creates new objects and relationships to represent, use and communicate design rationale for conflict mitigation in a collaborative environment. SHARED-DRIMS then keeps track of the intent and artifact evolution through the objects provided by some of the DICE modules and by the information provided by the designers. This combination allows the design rationale capture, even when pre-defined processes and products are changed by the designers. Figure 6-2 shows these modules and their interfaces. The modules are presented below with an emphasis on their functionality as used by SHARED-DRIMS:

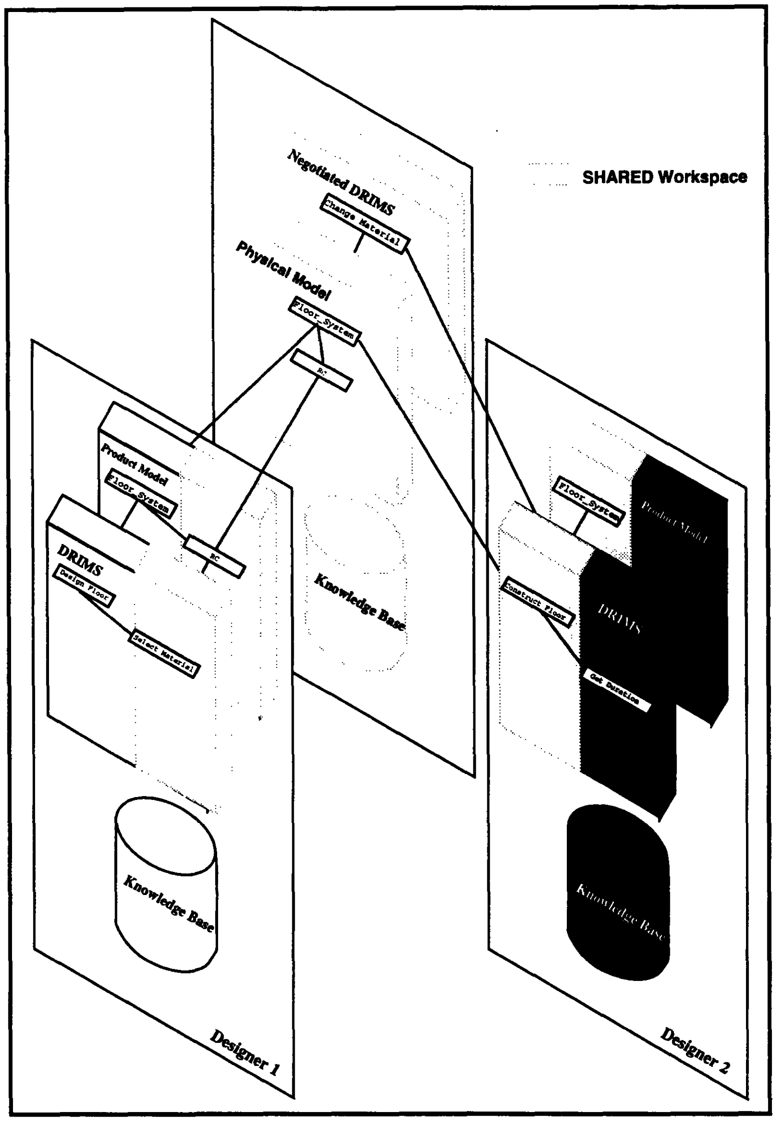


Figure 6-1: SHARED-DRIMS Conceptual Organization with two designers and a negotiation workspace.

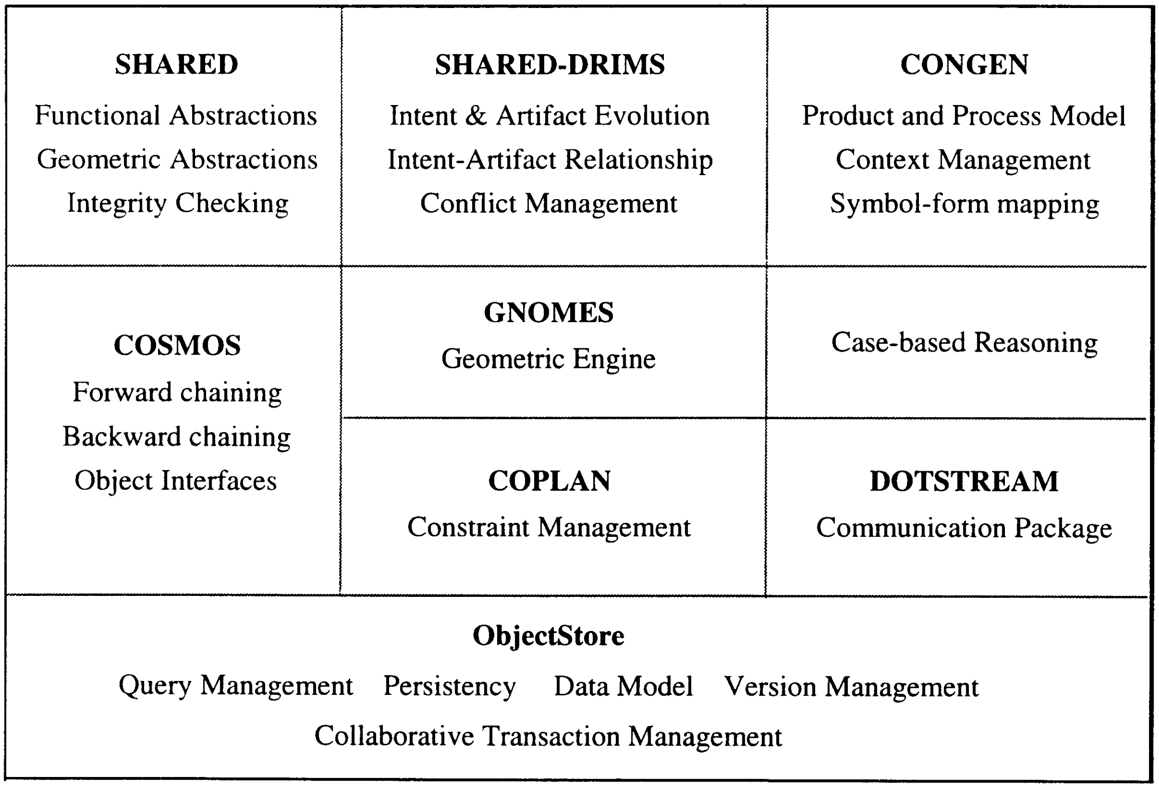


Figure 6-2: SHARED-DRIMS Architecture.

- $ObjectStore^{TM}$ [Object Design, Inc., 1991] provides persistent storage, query management, version management and collaborative transaction management. These functionalities are necessary for having access to a large number of objects representing the design rationale.

- *COSMOS* [Sriram et al., 1994] provides the inference mechanism for processing rules used to recommend the characteristics of an artifact or to recommend a design plan. COSMOS further provides the inference chain that led to a particular recommendation, that is, the rationale for providing such a recommendation.

- *COPLAN* [Fromont and Sriram, 1992] provides the constraint handling mechanism needed for keeping consistency of constraints introduced by intents, justifications, and recommendations.

- *DOTSTREAM* [Culbert, 1992] provides the communication mechanism for information transfer between different designers working in different places and at different times. This allows SHARED-DRIMS to support the communication between several designers working on a project.

- *GNOMES* [Sriram et al., 1991] provides geometric information of the evolving design. This facility is needed for keeping track of the geometry of the design related to a designer's rationale.

- *CBR*, when developed, will provide the mechanism for generating design alternatives from a set of design cases. This is necessary because designers sometimes select certain artifacts due to their similarities to previous cases that they have seen.

- *SHARED* [Wong and Sriram, 1993b] provides the representation of the functional and geometric abstractions of the artifact and maintains integrity in a collaborative environment. Designers work at different abstractions when trying to find an artifact that satisfies their intents. These abstractions need to be represented and expressed in a consistent language that could be shared across different disciplines.

- *CONGEN* [Gorti and Sriram, 1993] provide support for generating alternative designs from the combination of pre-defined "building blocks." This functionality captures the process by which different blocks are put together in order to create the artifact that meets the designers need.

### 6.2.2 Design Rationale Module

The design rationale module provides the link structure and templates for the information provided by design agents as their design rationale. These link structures provide the design rationale connection to the product as represented by SHARED system and component abstractions. These connections link the SHARED abstraction with the intent that they satisfy and the justifications that support the SHARED abstraction use. The design rationale also provides a mechanism for capturing the rationale from different sources such as design support systems, past designs and user's inputs. Moreover, the design rationale module is also responsible for using the captured information from this module and the base modules in conflict mitigation. In order to explain the functionalities of SHARED-DRIMS, an example is used throughout this chapter. This example provides the context in which the abstract concepts can be easily understood.

## 6.3 Design Rationale Capture

SHARED-DRIMS is designed to provide active support for generating design rationale. The system has to provide some of the design rationale without asking the designer for it.

This support may be provided in three forms: it uses the inference network provided by a design support system (CONGEN), it searches for similar recommendations produced in the past when designers overwrite design support system's recommendations, it accepts recommendations or justifications given by the designer.

### 6.3.1 CONGEN inference network

CONGEN provides the inference network for justifying a recommendation. CONGEN uses a backward and forward inference process. In this process rules encapsulates the actions to be taken related to an intent or to a particular characteristic of the artifact. In the case of a particular characteristic of the artifact, the rules are augmented so that the intent being sought by the rule is explicit. In the bridge example, the rule that says that if a span is less than 200 feet then a two span bridge is feasible, is seeking to minimize cost in addition to providing a solution for a support structure. The intent of minimizing cost is implicit in the rule. Thus, the rationale of selecting a two span bridge is to satisfy two intents. The first is the intent for a good transfer of loads and the second is the intent to minimize the cost of the bridge. Thus, these intents sought by that recommendation have to be made explicit.

CONGEN rules can be mapped to DRIM constructs. They have the designer (i.e., the particular knowledge base being used for making the decision), the recommendation (i.e. the *then part* of the rule), the intent (i.e. the other intents that the rule are seeking are put in *the comment part* of the rule), the justifications (i.e. the rule itself with the *if part* as the conditions that needs to hold true for the recommendation to be valid and satisfy the intent). Each clause of the precedent part of the rule is taken as a proposal that exists before asserting the effects in the *then part* of the rule. Each one of these proposals supports the presentation of the proposal made by the rule.

For example, a rule for selecting the material of the deck of the bridge can be mapped to DRIM constructs as follows.

Name: *timber_deck*

If *traffic_load* = *light* and *load* $\neq$ *abrasive*

Then *Deck.material* = *timber*

Comments: Maximize life of the bridge

$$
\begin{aligned}
t_1 \Rightarrow timber\_deck(select\_material, C) \quad &\Rightarrow \quad (P_1 \,\&\, P_2 \,\&\, S(P_1, P_{v1}) \,\&\, S(P_1, P_{v2}) \\
&\quad\;\; S(P_2, P_{v1}) \,\&\, S(P_2, P_{v2})) \qquad (6.1) \\
where \quad P_1 &= \{\{r : Deck.material = timber\}, \\
&\quad \{j : (\textbf{If } traffic\_load = light \\
&\quad \textbf{and } load \neq abrasive \\
&\quad \textbf{Then } Deck.material = timber), \\
&\quad (traffic\_load = light), \\
&\quad (load = light - weight)\}\} \\
and \quad P_2 &= \{\{r : \textbf{introduces}(Maximize\_Life)\}, \\
&\quad \{j : (\textbf{If } traffic\_load = light \\
&\quad \textbf{and } load \neq abrasive \\
&\quad \textbf{Then } Deck.material = timber), \\
&\quad (traffic\_load = light), \\
&\quad (load = light - weight)\}\}
\end{aligned}
$$

This rule introduces two proposals with a recommendation (*r:*) and a list of justifications (*j:*) at time $t_1$ within a context (*C*). It first introduces the use of timber for the deck. Then, it introduces the notion of maximizing the life of the bridge. This proposal sets a notion that any new material that should be considered should have a life span equal to or better than that of timber.[2] This proposal shows the increase in process detail. Intents that were not recognized at the beginning now emerge from the rules taken into consideration. In addition the rules present the support structure between the proposals that introduced *traffic_load = light* (that is $P_{v_1}$ where $v_1$ is *traffic_load*) and $load \neq abrasive$ (that is $P_{v_2}$ where $v_2$ is *load*) to the new proposals ( $P_1$ and $P_2$).

The previous paragraph presented the use of a rule as a rationale for a recommendation. However, a design support system like CONGEN may not provide a single rule rationale for a recommendation. It may provide an inference network as the design rationale. The inference network is produced from an inference performed on intent or artifact objects introduced earlier. These intent or artifact objects may be the results of a previous stage in the design. In this case, the rationale for its existence is assumed to exist.

Figure 6-3 shows the inference network for selecting the structural material of the bridge according to the following rules:

Name: *R4*

If *vehicles_per_day* $\leq$ *30*

Then *traffic_load = light*

Comments: Determine traffic load

[2] Timber and any other material referred here are objects with attributes and methods. In this case, the life span is an attribute of the material object that would be analyzed by the system when testing different materials.

Name: *R5*

If *location = tourist_area*

Then *load = II*

Comments: Determine type of load

Name: *R2*

If *traffic_load = light* and *width* $\leq$ *20 feet*

Then *Bridge.type = I*

Comments: Determine type of bridge

Name: *R3*

If *load = II Xor load = I*

Then *impact = low*

Comments:

Name: *R1*

If *Bridge.type = I* or *climate = mild* or *impact = low*

Then *Deck.material = timber*

Comments: Maximize life of the bridge

In this case, the inference uses other rules to derive values that are required to make the decision. These other rules introduce new proposals and make appropriate links to existing proposals. The rules and values that are connected through *and*, *or* , or *Xor* create support proposals between each of the predicates and the then part of the rule. In addition, the rules that are connected with an *Xor* create a contradict connection between the predicates of the rule. This is due to the characteristic of *Xor* that allows a rule to be valid only if one and only one of the predicates holds. This implies that the predicates are in competition and they cannot coexist in the same environment. Figure 6-4 presents the proposal structure built by the inference network. The translation of the network to the proposal structure is as follows:

$$R_4(i_4, C) \Rightarrow (P_{R4} \;\&\; P_{IR4} \;\&\; S(P_{R4}, P_{IR4}) \;\&\; S(P_{R4}, P_{V4})) \tag{6.2}$$

- Equation 6.2 explains that rule R4 presents two proposals $P_{R4}$ and $P_{IR4}$. $P_{R4}$ introduces the recommendation while $P_{IR4}$ introduces the intent sought by the rule. In addition, the rule makes supporting connections between these two proposals and then a supporting connection between $P_{R4}$ and $P_{V4}$ (proposal that introduced value 4). This last connection is due to the condition presented by $P_{V4}$ that fires R4.

$$R_3(i_3, C) \Rightarrow (P_{R3} \;\&\; S(P_{R3}, P_{V2}) \;\&S(P_{R3}, P_{R5}) \;\&\; C'(P_{R5}, P_{V2})) \tag{6.3}$$

- Equation 6.3 explains that rule R3 presents a proposal $P_{R3}$. $P_{R3}$ introduces the recommendation. In addition, the rule makes supporting connections between $P_{R3}$ and $P_{V2}$. This last connection is due to the use of a conditional presented by $P_{V2}$ that fires R3. However, R5 is also fired, providing $P_{R5}$ which according to the Xor condition cannot also be true. In this case, there is a contradicting relationship between $P_{R5}$ and $P_{V2}$.

The equations follow the same outline as the previous two, ending up with the recommendation provided by R1 which supports proposals from rule R2, R3 and V1. This set of equations shows that the system is able to take an inference network and parse it into a proposal network in which the recommendations, justifications, and intents are explicitly stated.

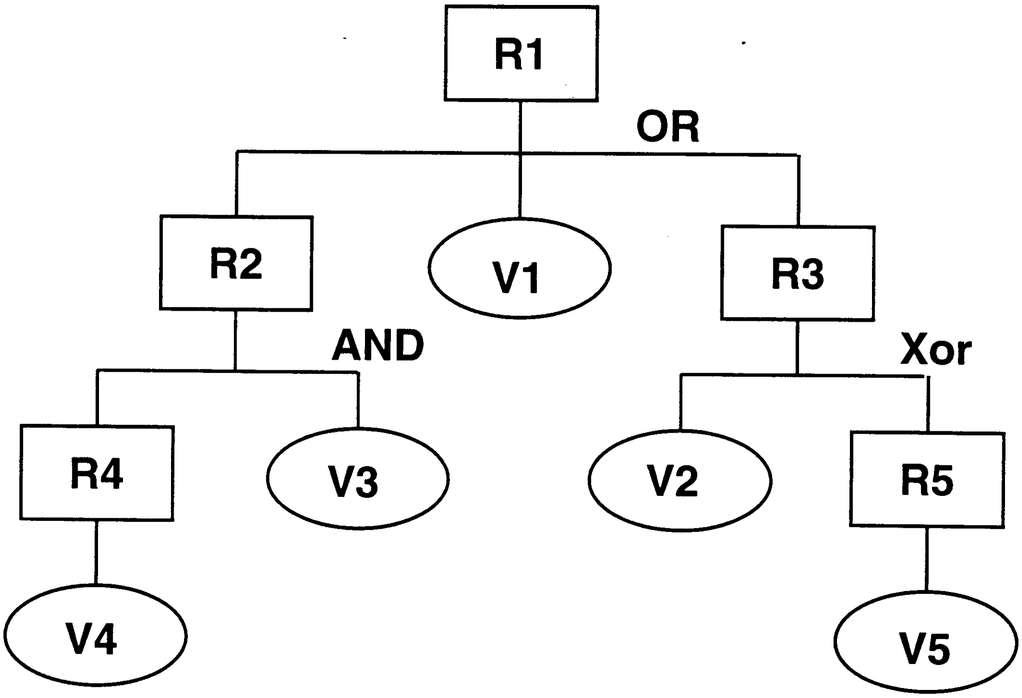


Figure 6-3: Inference Network.

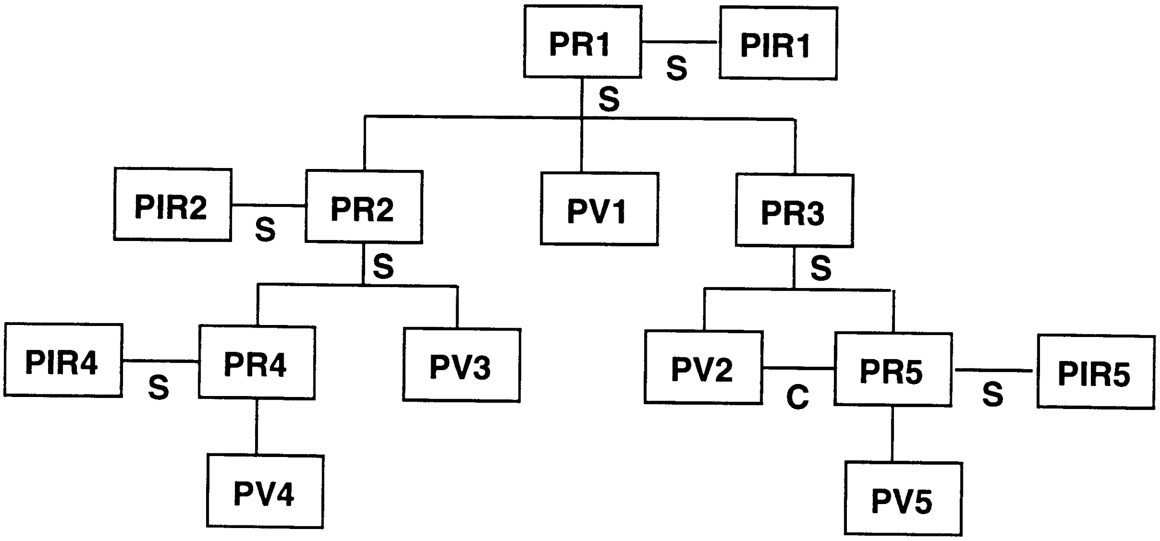


Figure 6-4: Proposal Network.

During design, the designer may receive several options given by the design support system. In such cases, the designer selects the recommendation. The next section presents the approach taken by SHARED-DRIMS when the designer selects one of the recommendations given by the design support system.

### 6.3.2 Similar Recommendations

Sometimes, the designer may receive several recommendations from the design support system. In such a case, the designer chooses one of the recommendations. However, that decision should also be recorded with its rationale. For this, SHARED-DRIMS searches the database for recommendations similar to the one made by the designer.

In order to retrieve the relevant recommendation from past design cases, the designer needs to make use of the context of the current design and previous designs. The problem is that the recommendations made in the past and the present recommendation may not be because of the same reason. In this case the system tries to isolate the similitude and extract the reasons for selecting such a recommendation. To do that, the system can take two approaches. First, it can find the extra intents that the chosen recommendation satisfies that were not taken into consideration by the design support system. Second, it investigates whether the rejected recommendations violate any implicit intents, which are not stated to the design support system. In the first case, the system assumes the following premise:

$$\exists pr \subset Pr, \forall pr_i \in pr, \exists r_\ell \subset R_{pr_i} : \forall r_{\ell_m} \in r_\ell \Rightarrow r_{\ell_m} = r \qquad (6.4)$$

where

- *pr* is a sub-set of projects.

- *Pr* is the set of all the projects in the database.
- $pr_i$ is a particular project from the *pr* sub-set.
- $r_\ell$ is a sub-set of recommendations.
- $R_{pr_i}$ is the set of all recommendations for project $pr_i$.
- $r_{\ell_m}$ is a particular recommendation from the $r_\ell$ sub-set that is equal to the recommendation *r* that was made by the designer and the system is looking for a rationale.
- Note that *pr* can be a null set. In this case there will not be any recommendation to search in the database.

Once the system has found the first project that satisfies the condition in Equation 6.4, the system creates a list of the intents related to that recommendation. Then, the rationale of that recommendation is sought according to the following criteria:

$$\exists in \subset I_{r_{\ell_m}} : \forall in_q \in in \Rightarrow in_q \notin I_r \tag{6.5}$$

where

- *in* is a sub-set of intents.
- $I_{r_{\ell_m}}$ is the set of all the intents related to the recommendation $r_{\ell_m}$ in the project $pr_i$.
- $I_r$ is a the set of all the intents related to the recommendation *r* to which the rationale is being sought.
- $in_q$ is a particular intent from the *in* sub-set that is not in the set $I_r$.

Equation 6.5 finds all the intents that were not taken in consideration by the design support system. This is due to the introduction of new intents during the process that were not planned in the design support systems. This inclusion of new intents allows SHARED-DRIMS the flexibility to cope with the increase in detail in the projects and the changing conditions from project to project. Once those intents are found, the system tries to validate them in the new context according to the following criteria:

$$\exists inte \subset in : \forall inte_q \in inte, \exists in_{inte_q} : in_{inte_q} = \textbf{Intent}(inte_q) \; \& \; in_{inte_q} \in I_{pr_r} \qquad (6.6)$$

where

- *inte* is a sub-set of intents.
- $inte_q$ is a particular intent from the *inte* sub-set.
- $I_{pr_r}$ is the set of all the intents in the project of the recommendation to which the rationale is being sought.
- $in_{(inte_q)}$ is the intent that introduced $inte_q$ and is in $I_{pr_r}$.

Equation 6.6 selects all the intents related to the intent that introduced the recommendation in the old project that is similar to the one that the system is searching for the rationale (new project). That parent intent should be in the context of the new project. If it is, then the system tries to build the relationship between the intent and the recommendations without rationale (new project). This is performed by checking the justification and finding out if they are valid in the new context according to the following criteria:

$$\exists inten \subset in_{(inte_q)} : \exists d_t \in D_{p_{inten}, r_{\ell_m}}, d_t(inten, C) \Rightarrow p_r \qquad (6.7)$$

where

- *inten* is a sub-set of $in_{(inte_q)}$ in Equation 6.6.
- $p_{inten,r_{\ell_m}}$ is the proposal that associates the $inten$ and the recommendation $r_{\ell_m}$ from the previous project and presented in Equation 6.4
- $d_t$ is a particular design operator or justification used in the proposal that introduced recommendation $r_{\ell_m}$. $d_t$ can be applied using a design support system and testing the design operator. In the case of CONGEN, the system will fire the rule and test that it does test true.
- $D_{p_{inten,r_{\ell_m}}}$ is the set of all the design operators or justifications used in the proposal $p_{inten,r_{\ell_m}}$.
- $p_r$ is a new proposal that supports the recommendation to which the rationale is sought.

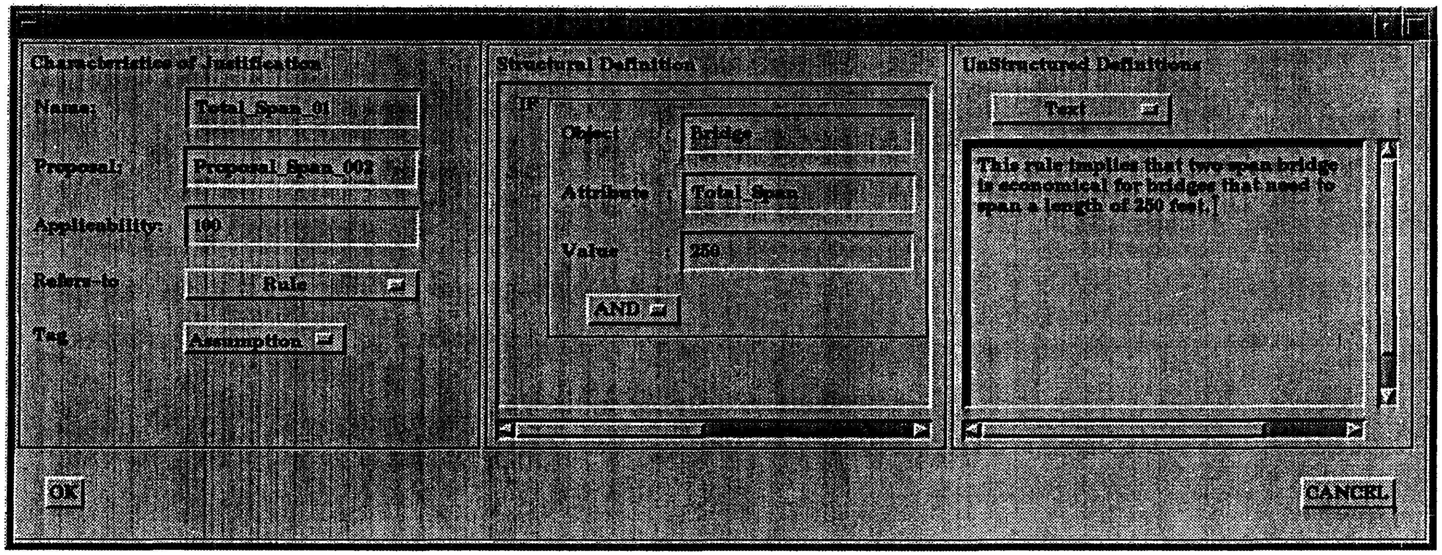
The results from Equation 6.7 are presented to the designer as a reason for selecting the recommendation. The system has two modes by which it searches the rationale of alternative selected by the designer. These two options are lazy and greedy searches. The lazy search interacts with the designer looking for rationale until the designer is satisfied. In the greedy search the system does an exhaustive search; i.e, the system will continue searching all the intents, and projects until there is no more rationale or justification for selecting a given alternative. There is a trade-off between the two. The lazy one may find some of the reasons for a given alternative but not all, while the greedy search is time consuming. Thus, the designers are left to decide which strategy is best for them according to the project needs.

### 6.3.3 User input

The previous section shows the process by which the rationale is captured when the designer has to make a decision about different alternatives presented by the design support system. Other times, the user needs to introduce a recommendation that was not taken into consideration by the system. In such cases the system allows the user to use the constructs provided by SHARED-DRIMS to provide the information asked. Each item asked can be given in two ways. One is through a structured template, in which the user identifies the structure of the answer and enters data in the appropriate slot. The other is to enter free text information (see Figure 6-5).

This dual representation allows for encapsulation of both human and computer rationale. The advantage of one over the other is that structured information can be reasoned about but unstructured or free text cannot. The trade-off is support for detecting conflicts or support for recording the rationale. In both cases, the rationale is going to be recorded. In the first case, the information is going to be used by the humans and will have to be stored for browsing. In the other case, the computer can make use of the information since the information is in a form that can be used for inferencing. The computer can then use the information for building assumption relationships between different designers, and to detect conflicts at the physical level when the conditions used are different from the conditions already existing in the physical model of the artifact.

If the designer uses structured information to record the recommendation, then the system can help the designer in recording the rationale following two processes. First, it finds if the recommendation has been made in the current project but was rejected due to some rationale. That rationale is presented to the designer so that he/she can modify it or change his/her mind. The second process is similar to the process presented in



Characteristics of Justification

Name: Total_Span_01

Proposal: Proposal_Span_002

Applicability: 100

Refers-to: Rule

Tag: Assumption

Structural Definition

IF

Object : Bridge

Attribute : Total_Span

Value : 250

AND

UnStructured Definitions

Text

This rule implies that two span bridge is economical for bridges that need to span a length of 250 feet.

Figure 6-5: SHARED-DRIMS: Design Intent Window with both structured and unstructured representation.

Section 6.3.2. The system would search other projects for similar recommendation and present the rationale of the recommendation that is valid in this project.

If the designer chooses to record an unstructured rationale or recommendation, the system files the proposal with the information but cannot use it for retrieving other rationale or using it for conflict detection. The system can only use what it can understand. However, the value of that information may deem it to be important for the designer when looking at the history of the project.

## 6.4 Conflict Mitigation

Once the information about design rationale is recorded and represented through DRIM, the system uses that information for conflict mitigation. Conflict mitigation is divided into rationale dependencies, conflict detection, conflict causes, conflict resolution, conflict negotiation, solution impact and conflict prevention. This is based on the hypothesis that conflicts can be mitigated by the following four steps: 1) making the dependencies between the rationale of the different designers explicit, 2) detecting any inconsistencies between past information and any new information; 3) presenting the reasons for the inconsistent values to the designers; 4) using any knowledge in the computer to find a solution to the conflict; 5) allowing the designers to use the reasons for resolving the conflict; 6) using any knowledge (in the computers or designers) to determine the impact of the solutions as well as prevent more conflicts (see Figure 6-6). The following section explains each one of them in more detail.

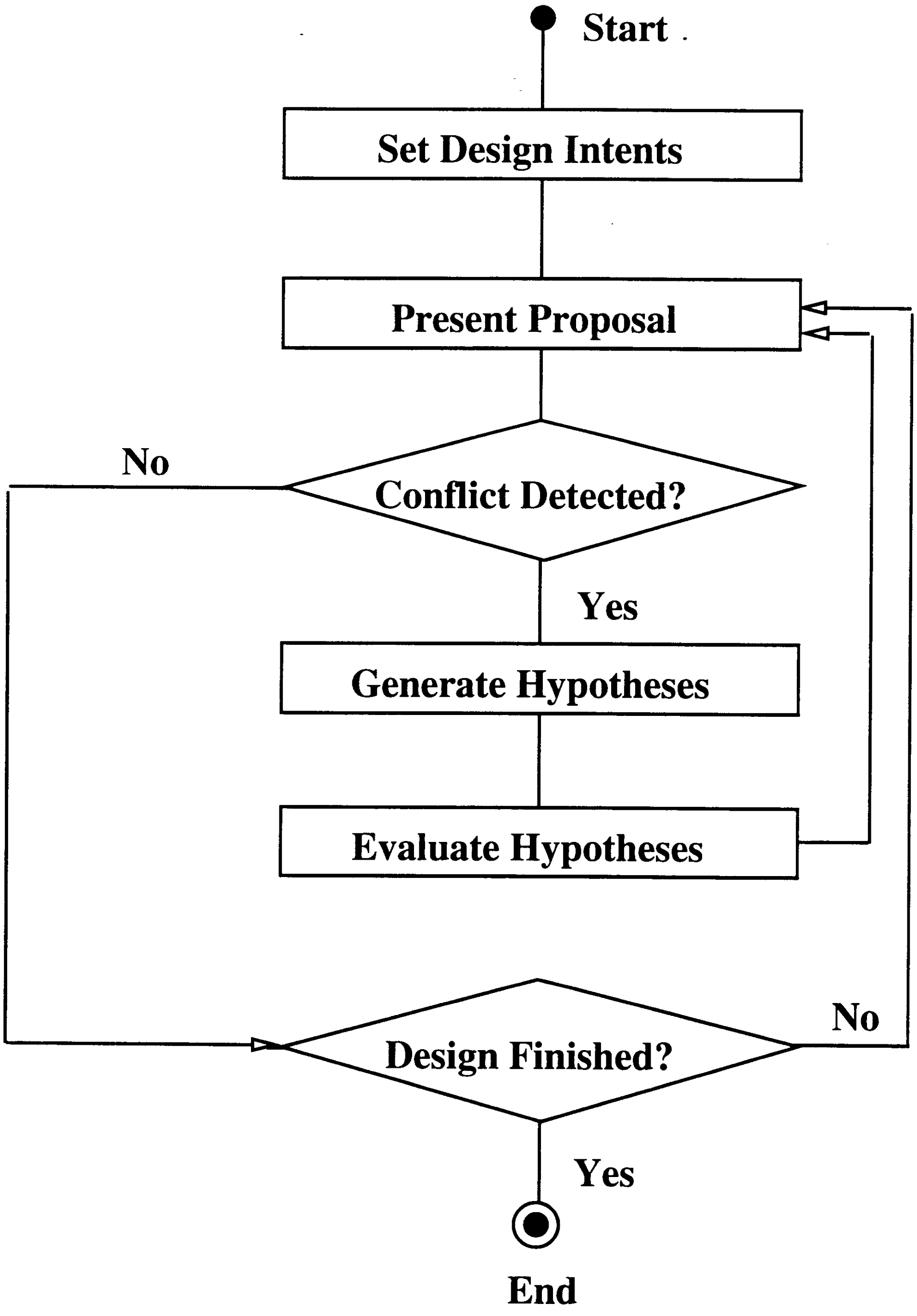


Figure 6-6: Conflict Mitigation.

### 6.4.1 Rationale Dependencies

Designers may use information which interacts with other pieces of information, either from the same design domain or from another design domain. There are cases in which a design variable is defined by different designers. This variable may be modified by all the designers. Such a variable is called shared responsibility. There are other cases in which a design variable is defined by only one designer. This variable may be used by all the designers but they cannot modify it. Such a variable is called producer-consumer responsibility. Both of these variables are sources of conflicts but they differ in how responsibility for the resolution of the conflict is assigned. In the shared responsibility all designers are responsible for changing the variable until an agreement has been reached. In the producer-consumer responsibility, the producer is responsible for changing the variable taking into consideration the consumer issues.

There are two type of variables that may cause conflicts: shared responsibility and producer-consumer responsibility. These variables have different dependencies among them. The dependencies of shared responsibility variables can be of three types: 1) the design parameter serves multiple purposes and is set from different perspectives; 2) two different objects intersect partially or completely, violating physical constraints; and 3) there are different parameters but they use the same resources. The dependencies of producer-consumer responsibility variables can be of two types: 1) a consumer makes an assumption about a variable that is going to be made by the producer, and 2) the consumer uses a variable made by the producer.

In each case SHARED-DRIMS sets the dependencies between the different elements of the design. It builds the dependencies between proposals in conflict. These conflicts occur by the proposals introduction of recommendations that may cause shared or producer-consumer responsibility type of conflict. These dependencies are through the *reacts-to* links between proposals. SHARED-DRIMS achieves these tasks while capturing the design rationale through the use of DRIM (see Figure 5-1).

### 6.4.2 Conflict Detection

This research makes the assumption that design intents do not conflict and that only their implementation does. Thus, SHARED-DRIMS detects conflicts through the recommendations made in the proposal. The conflict detection depends on the type[3] of conflicting variables. In the case of shared responsibility parameters with dual purpose, the system maintains a global repository of recommendations which checks if a recommendation is already in the repository. Thus, the system is able to test if a recommendation already exists, what is the value of this recommendation, what is the range in which the value can change without causing conflicts and who are the designers involved in setting the value of this design parameter. This type of conflict is the only one that SHARED-DRIMS can detect by itself. The other types of conflicts are detected by other DICE systems, which interact with SHARED-DRIMS. As far as the implementation, it is assumed that the DICE sub-systems can detect the conflict and provide the violating recommendations to SHARED-DRIMS. In the current implementation of SHARED-DRIMS, these systems are not fully integrated and their input/output is simulated.

In the case of shared responsibility for intersecting objects, SHARED-DRIMS relies on SHARED [Wong and Sriram, 1993b] for detecting physical interactions and signaling the objects that are in conflict. SHARED-DRIMS takes the proposals that introduce the recommendation and presents them to the designers in conflict. SHARED uses a

[3] shared or producer-consumer responsibility type of variables

distributed rule base mechanism. For example, an environmental engineer may reject the use of a central pier of a bridge as recommended by a structural engineer, the environmental engineer may introduce a constraint in the form of a rule that stipulates that there be no pier inside the river.

SHARED-DRIMS deals with conflicts due to resource allocation by maintaining knowledge of the usage and the availability of resources during the design plan. This kind of conflict is detected by the SCHEREC system [Pena Mora et al., 1994] for activity planning.

In the case of producer-consumer responsibility assumption variables, SHARED-DRIMS searches the repository in order to determine whether a decision has been suggested or assumed for a given recommendation. However, for the case of using information from the producer and spreading it through the design, SHARED-DRIMS uses several techniques depending on how the relationship originates. In the case of quantitative relationships –through numerical equations and inequalities– SHARED-DRIMS relies on COPLAN for resolving the constraints and propagating them. In the case of qualitative dependencies, SHARED-DRIMS relies on CONGEN to perform qualitative reasoning in order to determine the dependencies and the impact of change. For example, a geotechnical engineer recommends the use of piers perpendicular to the terrain because his/her studies show that the terrain can only support load perpendicular to it. Thus, the geotechnical engineer inputs into COPLAN the terrain as a set of equations that represent its slopes. Then, he/she inputs the constraint that a pier must have a slope that is normal to the slope of the terrain.

```
Constraint(slope-bank1, (if ($x >0 & $x <100), sb = -0.5773))
Constraint(slope-bank2, (if ($x >150 & $x <250), sb = 0.5773))
Constraint(slope-bed, (if ($x >150 & $x <250), sb = 0))
Constraint(slope-pier, ( ss = ($y1 -$y)/($x1 -$x)))
Constraint(intersec, ( (sb * ss) = -1 ))
```

In the case that the relationship was made by using heuristics, SHARED-DRIMS relies on COSMOS for maintaining the inference network. In the case that the relationship is made by drawing from cases, the system relies on the case-based reasoner that is under development. If the designer enters non-electronic catalog information as a justification for a recommendation, the relationship is maintained and propagated with the help of the designer. However, if the catalog is electronic, the relationship is maintained and propagated with the help of SHARED-DRIMS searching mechanisms through a catalog repository. In the case of authority and prototypes, SHARED-DRIMS interacts with the designer to propagate the effect of changes since the system does not have information on how to relate such elements.

Thus, conflicts are detected by the mechanism used for the generation of proposal relationships. SHARED-DRIMS uses recommendation repository for detecting recommendation changes as well as other DICE systems for other kinds of conflicts. However, when a conflict is resolved by changing the value of one of the variables, the new value needs to be propagated. In SHARED-DRIMS, a recommendation can be made using different types of reasoning. Thus, there is the possibility that a new value may be useful in various types of reasoning different from the reasoning that used the old value. For example, if a value allowed a heuristic to be applied and later it is changed, then the new value may cause a new heuristic or a case to be applied as a justification for a proposal. In these cases, careful attention has to be given to a conflict propagation mechanism. These issues will be considered in a future project.

### 6.4.3 Conflict Causes

In a collaborative environment, the designers are drawn from different domains and they work together. However, this collaboration does not mean that they understand each other completely. They have some common knowledge. That common knowledge is the one that will help to resolve the conflict.

This paper is based on the belief that the collaborating agent will try to incorporate conflicting concerns missing from different proposals into account during his/her design. The system makes those conflicting concerns explicit and presents them clearly to the involved professionals. However, for the designer to take those conflicting concerns into consideration, he/she must understand them. For this reason, the system tries to find a common language between the professionals (see Figure 6-7). It searches for notions about the professional intents and justifications in the other conflicting domains. This process checks each one of the justifications that support the recommendation in conflict, as well as for all the proposals that support the conflict proposal; the conflict proposal is the one which proposes the recommendation in conflict. SHARED-DRIMS searches if the justification exists in the other professional domains. If it does, then it incorporates that as an intent to be sought. If not, the system looks at the intent. If the intent is not understood (not found in the domain), the system searches the proposals that introduced the intent. In this case, the intent becomes the recommendation sought to be understood.[4] Thus, the justifications and intents of the intent are looked up until an intent is understood or when the search has been exhausted.

In the case that the search has been exhausted, the system asks the designer for the

[4] Intents are also recommendations made at higher level of abstractions.

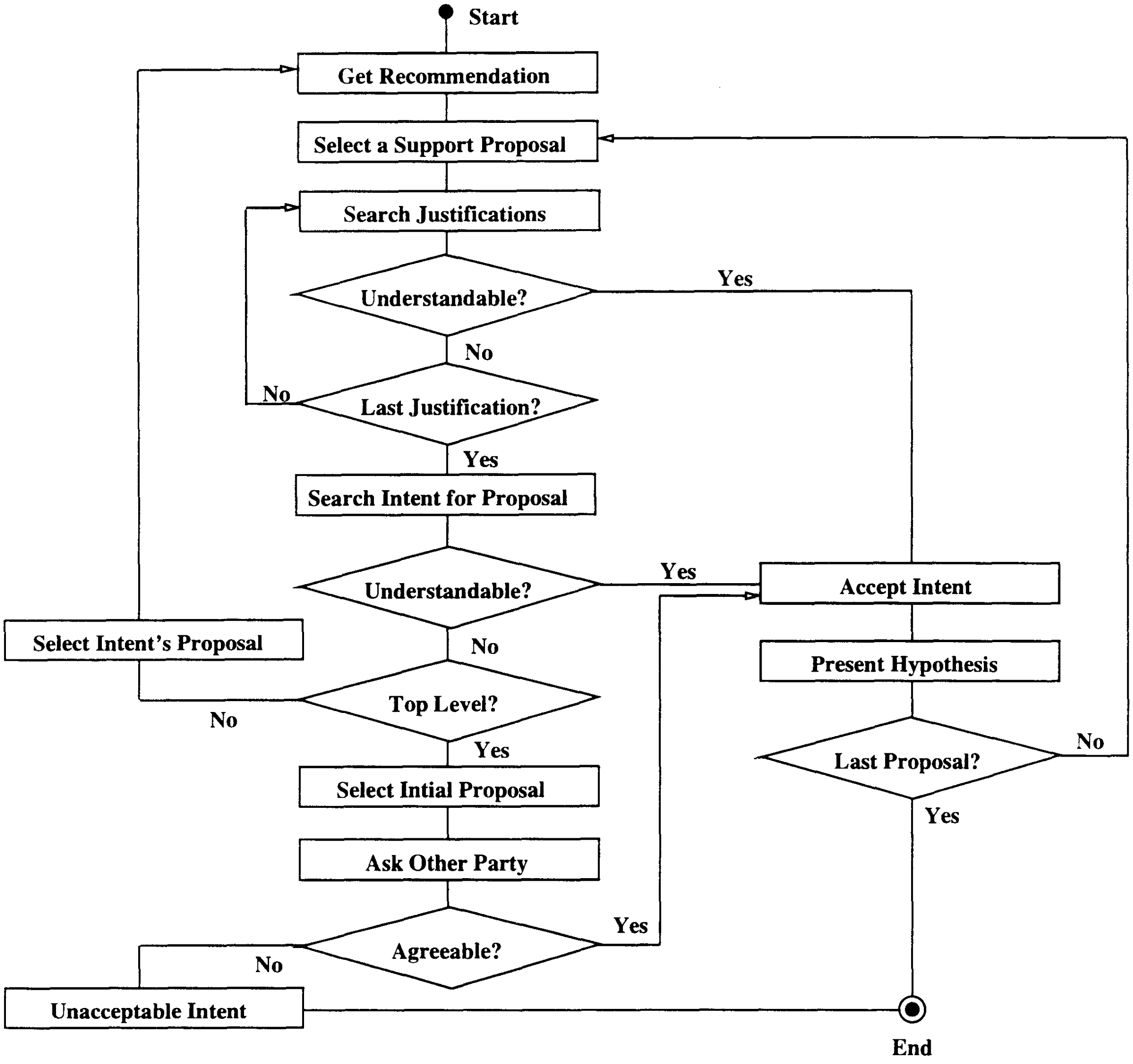


Figure 6-7: Hypothesis Generation.

intents from the lowest level. That is, it starts from the intent that introduces the conflicting recommendation and goes up until the user understands one of them. If there is no intent that is understood, then the conflicting professionals are made aware that they need to meet to resolve the differences.

### 6.4.4 Conflict Resolution

SHARED-DRIMS may use three strategies to resolve conflicts. First, the system may use knowledge in the form of heuristics for common conflicts. Second, SHARED-DRIMS may look at past designs for conflict resolution. Third, the system provides information for the human designers to efficiently resolve the conflict. In the first case, SHARED-DRIMS uses the COSMOS inference engine for resolving the conflict. In the second case, it tries to match the conditions from the past with the current conditions. In this case the notion of versions are really important. Versions capture the state of the design at any given point. Thus, by capturing the different versions of a conflict resolution, each stage of the resolution can be used as key entries for new conflicts (see Figure 6-8). Thus, in the case that a conflict arises, it may be one of the many stages of an old process. However, when the conflict cannot be found from previous design and the solution is not available, the third case is applicable. The system provides the designer with the information about the causes of the conflict so that they can negotiate accordingly. Then, the negotiation is performed using the constructs provided by the system.

### 6.4.5 Conflict Prevention

This task is achieved in conjunction with dependency maintenance. The system forecasts impact of a change so that the designers are aware of other conflicts which may be caused. Also it is used for presenting the impact of their decisions. In this case, during

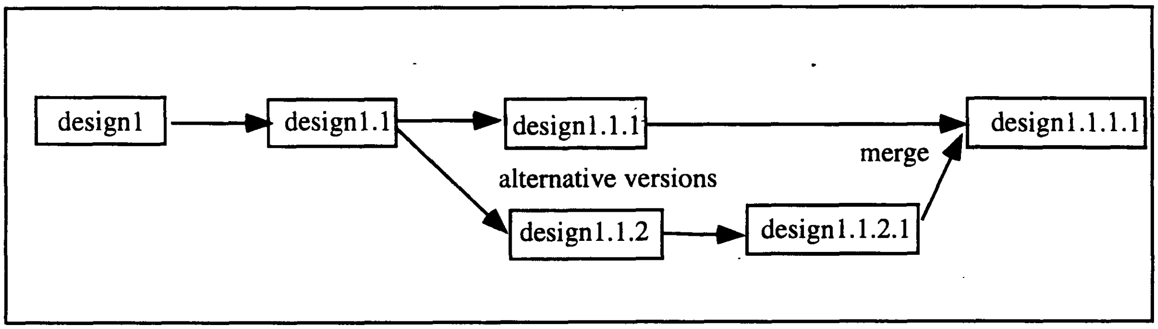


Figure 6-8: Versions for resolving conflicts with design1 - each version of the conflict resolution is a entry key for the graph.

conflict resolution and negotiation the system both presents the rationale of the conflicting recommendation, why it is there, and also its impact, what is going to happen if the value changes. This case is again very important and difficult because of the multiple types of dependencies that have to be maintained. These dependencies are sometimes discrete (without known pattern) where extrapolation is not possible unless knowledge about the new information is available.

## 6.5 Chapter Conclusions

In this chapter, the functionalities of a proactive project information management system (SHARED-DRIMS) have been presented. Among these functionalities are: 1) capturing rationale information from design support systems, past projects, and the user; 2) mitigating conflict by creating variable dependencies, inconsistency detection, variable rationale, conflict resolution, and conflict prevention. SHARED-DRIMS provides some of these functionalities, while the others DICE sub-systems provide the rest of the functionalities.

# Chapter 7

# Illustrative Example

*Example is the school of mankind, and they will learn at no other.*

Edmund Burke, *Letters on a Regicide Peace, letter 1, 1976*

## 7.1 Chapter Introduction

Section 2.2.5 presents the importance of having access to the reasons and the basis for a decision. It argues that a system that represents and manages both the reasons and basis for a decision would be able to improve the design-construction process. This improvement would come in terms of shorter process time and better quality. To prove this point, Section 7.2 presents an example of designing a bridge with SHARED-DRIMS. Section 7.2.1 shows the use of the system from a single participant point of view. Section 7.2.2 presents the system as used by multiple participants in the design of the bridge. Section 7.3 provides details about the software and hardware used for developing the system. Finally, Section 7.4 presents a summary of the chapter.

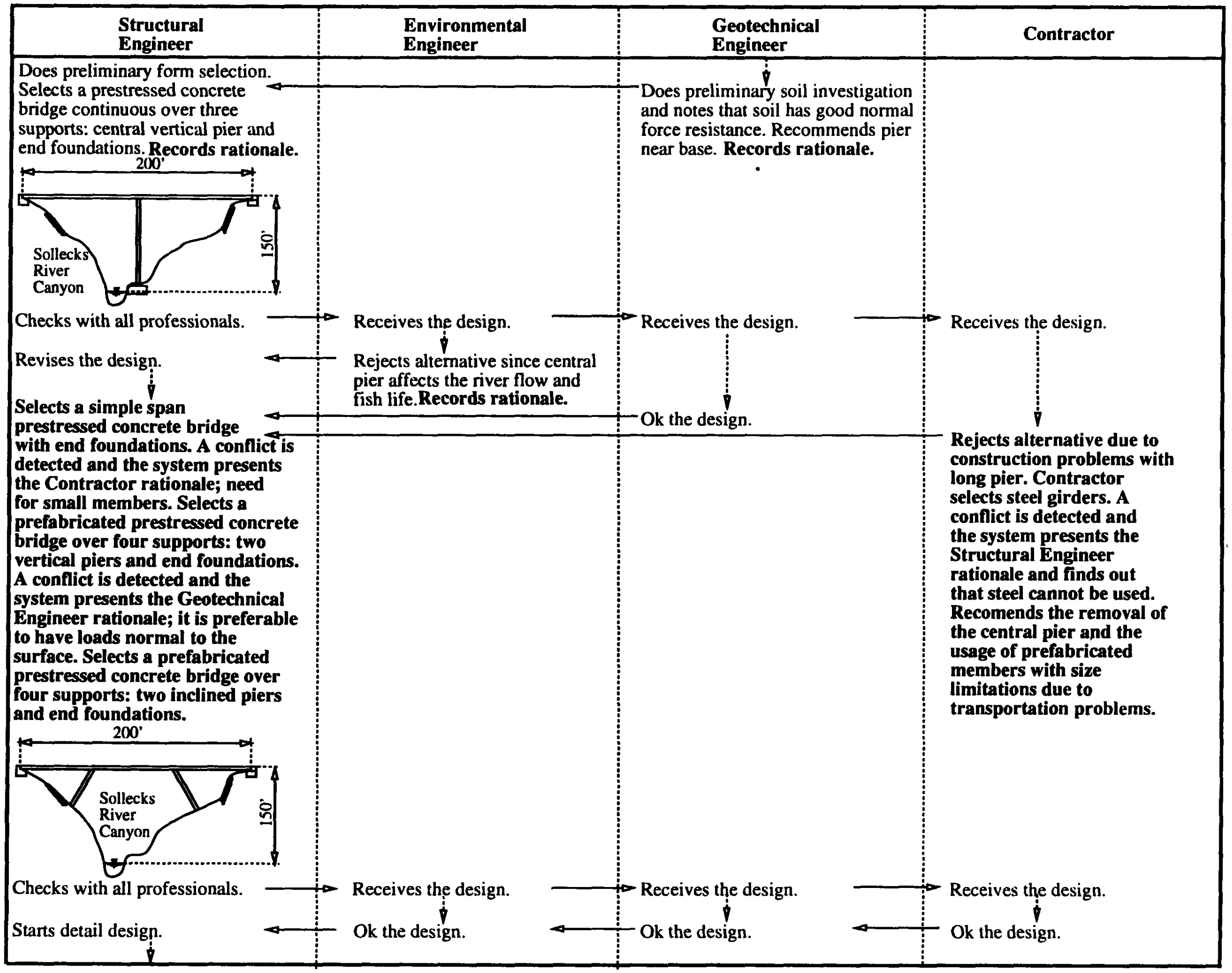
## 7.2 Bridge Design

Using SHARED-DRIMS, problems stated in Section 2.2.5 with the original design process may be obviated. SHARED-DRIMS is used by a structural engineer and three other professionals to reduce the number of interactions from five that this process took to two (see Figure 7-1) – this early identification and resolution of conflicts is conflict mitigation. This shorter process is described below: the geotechnical engineer records his rationale for recommending that the base of any pier be normal to the slope, which is based on the maximum resistance of the soil is to normal forces. The structural engineer records her rationale when deciding prestressed concrete and a two span bridge.

### 7.2.1 Single Designer Design Rationale Capture

In order to record her rationale, the structural engineer uses her top level intent (to design a bridge) to generate a proposal. That proposal has as recommendation the intent of safely withstanding the loads acting on the structure during its lifetime. The justification given by the structural engineer is the precepts of the profession (which is an authority type of justification in this case). Since the structural engineer works with SHARED-DRIMS in designing the bridge, SHARED-DRIMS in turn presents a proposal for designing the bridge. This proposal has as a recommendation a plan of action that consists of four goals: select the material, select the form, size the elements, and determine the loads (see Figure 7-3 for a C++ representation of a generic recommendation). The justification given by SHARED-DRIMS for such recommendation is a rule stored in its knowledge base (see Figure 7-2 and Figure 7-4).

After the plan has been established, the structural engineer allows SHARED-DRIMS



| Structural Engineer | Environmental Engineer | Geotechnical Engineer | Contractor |
|---|---|---|---|
| Does preliminary form selection. Selects a prestressed concrete bridge continuous over three supports: central vertical pier and end foundations. **Records rationale.** | | Does preliminary soil investigation and notes that soil has good normal force resistance. Recommends pier near base. **Records rationale.** | |
| Checks with all professionals. | Receives the design. | Receives the design. | Receives the design. |
| Revises the design. | Rejects alternative since central pier affects the river flow and fish life. **Records rationale.** | | |
| **Selects a simple span prestressed concrete bridge with end foundations. A conflict is detected and the system presents the Contractor rationale; need for small members. Selects a prefabricated prestressed concrete bridge over four supports: two vertical piers and end foundations. A conflict is detected and the system presents the Geotechnical Engineer rationale; it is preferable to have loads normal to the surface. Selects a prefabricated prestressed concrete bridge over four supports: two inclined piers and end foundations.** | | Ok the design. | **Rejects alternative due to construction problems with long pier. Contractor selects steel girders. A conflict is detected and the system presents the Structural Engineer rationale and finds out that steel cannot be used. Recomends the removal of the central pier and the usage of prefabricated members with size limitations due to transportation problems.** |
| Checks with all professionals. | Receives the design. | Receives the design. | Receives the design. |
| Starts detail design. | Ok the design. | Ok the design. | Ok the design. |

Figure 7-1: Improved process for selecting the form and material of the Sollecks Bridge

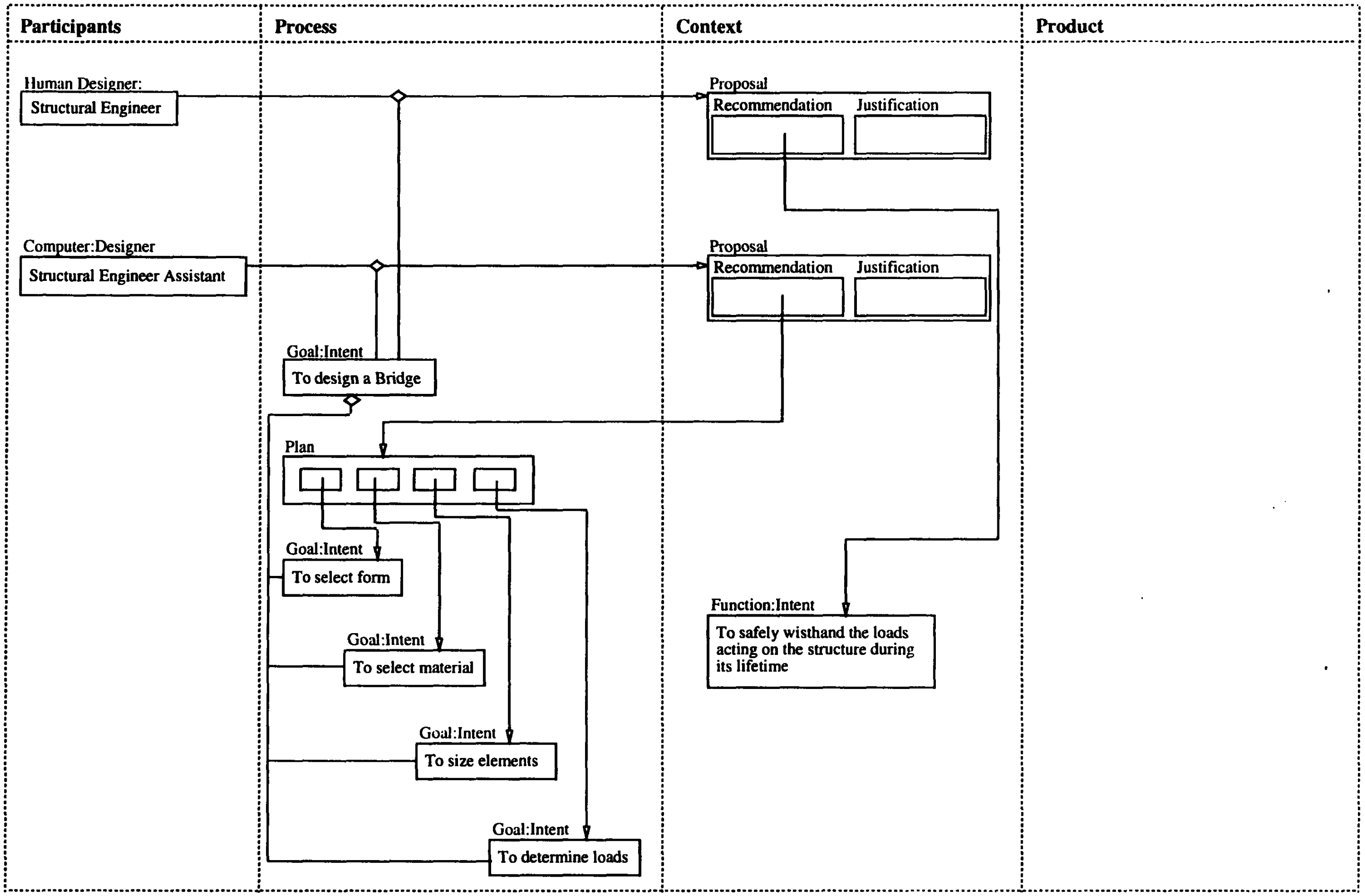


Figure 7-2: Initial proposals presented by the structural engineer and SHARED-DRIMS

```
 class DRIMIntroRecomm :   public DRIMObj, public COS_root {

private:

protected:
 GNlist<DRIMProposal*> modifying_proposals; /** The list of proposals that modified the recommendation**/
 GNlist<DRIMConstraint*> constraints; /** The constraint list **/
 DRIMProposal* proposal; /** The parent proposal **/

public:

 DRIMIntroRecomm ( char *, DRIMProposal *);
 DRIMIntroRecomm ( char *);
 ~DRIMIntroRecomm();
 virtual DRBoolean same(DRIMIntroRecomm*){ };
 virtual void  set_recommendation(DRIMRecommendation *reco){ recommendation = reco;}
 DRIMProposal *get_proposal();
 void set_proposal(DRIMProposal *prop);
 void insert_mod_prop(DRIMProposal *);
 os_List<DRIMProposal*> get_mod_prop();
 void check_constraint();
 void insert_constraint(DRIMConstraint *);
 os_List<DRIMConstraint*> get_constraints();
 void run_ie(char* rule_file, os_Set<COS_root*>* objects=NULL);
 /** access functions for cosmos inference engine **/
 virtual const char* get_classname(){return (Dis_a()->get_class_name());}
 virtual DBoolean super_class(char* c){return (Dis_a()->superclass(c));}
 /** the next 3 must be redefined in classes with new attributes and methods
   should call parent class methods **/
 virtual int put_value(char*,char*);
 virtual char* get_value(char*,char*);
 virtual DBoolean invoke_method(char*);
 DDECLAREEXTENT(DRIMIntroRecomm);
 DDECLARECLASS(DRIMIntroRecomm);
};
```

Figure 7-3: C++ code representation of a generic recommendation

```
(RULE: design_bridge 20
IF
(CLASS: DRIMIntent OBJ: $a
((((((
((refer_to == "Goal") AND (refer_to == $re))
AND (action == "Design")) AND (action == $ac))
AND (object == "Bridge")) AND (object == $ob))
AND (modifier == "Structural")) AND (modifier == $mo)))
THEN (
(EXECUTE OBJ: $a notify[Intent_Objects_Created])
(MAKE (CLASS:DRIMIntent OBJ:Create_Bridge (na "Bridge")
          (agent "Structural_Engineer") (refer_to "Goal") (action "Create")
          (object "Bridge") (modifier "Structural")
          (rule "design_bridge")
          (des "If_the_goal")
          (in $a)
          (ca $a) ))
(MAKE (CLASS:DRIMIntent OBJ:Select_Material (na "Bridge")
          (agent "Structural_Engineer") (refer_to "Goal") (action "Select")
          (object "Material") (modifier "Bridge")
          (rule "design_bridge")
          (des "If_the_goal")
          (in $a)
          (ca $a) ))
(MAKE (CLASS:DRIMIntent OBJ:Select_Form (na "Bridge")
          (agent "Structural_Engineer") (refer_to "Goal") (action "Select")
          (object "Form") (modifier "Bridge")
          (rule "design_bridge")
          (des "If_the_goal")
          (in $a)
          (ca $a) ))
(PRINT "Creating Intents" | $a | "associations\n")
)COMMENT: "")
```

Figure 7-4: Rule for designing a bridge

to proceed with the design under her supervision. In this case, SHARED-DRIMS suggests the creation of a place holder for the bridge. This place holder is just a generic artifact without any particular structure. The structure will be defined as the design proceeds and more decisions of the bridge structure has been made. Actually, the definition of the bridge structure is the next step to be followed by SHARED-DRIMS. SHARED-DRIMS presents proposals for the selection of the material of the bridge. The recommendation of the proposal is to use timber deck and girders. The justification of the proposal is a rule that says if the span of the bridge is less than 250 feet, then the use of timber is a feasible solution (see Figure 7-5). However, the structural engineer knows that timber in this locality can be a problem. So, she presents another proposal in which the recommendation is to avoid the use of timber. The proposal is based on the intent to maximize the life of the bridge. This intent was implicit until the point in which it takes prominence. The justification of the proposal is that timber would wear off under abrasive action of gravel and heavy loads. This interaction between a presentation of a proposal and its contradiction can be combined in one schema as shown in Figure 7-5. The two intents are combined and a contradiction link is built around the proposal. Finally, the justification has an adjective before it that says that the justification is of the contradicting type.

Once the proposal to use timber for the deck and girders is rejected. The structural engineer presents another proposal. This one has as a recommendation the use of concrete deck and timber girders. In reality, this proposal should be divided into two proposals that satisfy the same intent of selecting the material. Here the two proposals are combined because they satisfy the same intent and have the same contradicting justification (see Figure 7-6). The proposal is rejected due to difficulty of fastening the two materials. This contradiction is related to the intent of safely withstanding the loads acting on the structure. Thus, a new version of the old proposal is created. This time the proposal recommends

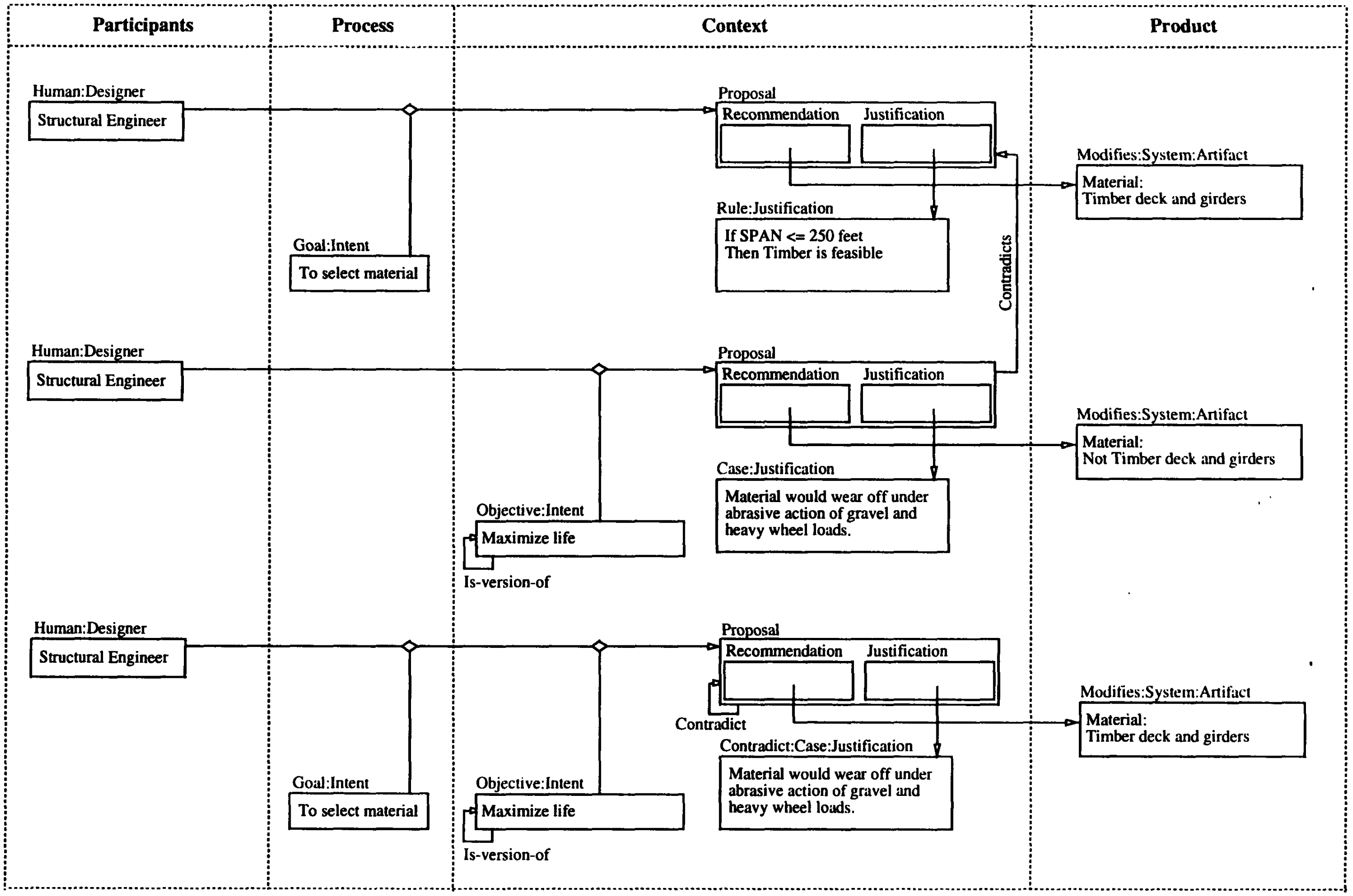


Figure 7-5: Proposal and contradicting proposal combined

the use of concrete deck and steel girders. This proposal is again rejected because it does not satisfy the intent of minimizing maintenance. This intent was implicit in the process until the decision was made to use steel. This shows that intents are not always set forth at the beginning but may be brought to the surface due to a set of decisions already made. Finally, a new proposal is introduced. This proposal has as a recommendation the use of prestressed concrete for both the deck and the girders. The justification is that this proposal satisfy all the other intents including the intent of minimizing maintenance.

Once the proposal of using prestressed concrete as material is accepted, the bridge artifact contains that new structure (see Figure 7-7). At this point, the bridge artifact has gone through an evolution from being just a place holder at first (see Figure 7-8), then to have only a material characteristic (see Figure 7-9), to have a structure of different parts (see Figure 7-10). This evolution creates a version tree of the bridge artifact (see Figure 7-11). Now, the structural engineer assistant, SHARED-DRIMS in this case, proceeds to satisfy the intent of selecting the bridge structural form. Figure 7-12 presents the proposal presented by SHARED-DRIMS about the form of the bridge. The recommendation is the use of a two-span continuous slab bridge over three supports. The justifications were a rule in SHARED-DRIMS knowledge base (see Figure 7-13) and a report presented by the geotechnical engineer which recommended the use of piers perpendicular to the terrain.

### 7.2.2 Multiple participants conflict mitigation

After the structural engineer and SHARED-DRIMS finish performing their designs, SHARED-DRIMS presents to all participants in the project (see Figure 7-14) the structural engineer design process. The system shows the structural engineer's view of the artifact hierarchy and a rough sketch of this design (see Figure 7-15). The bridge consists of

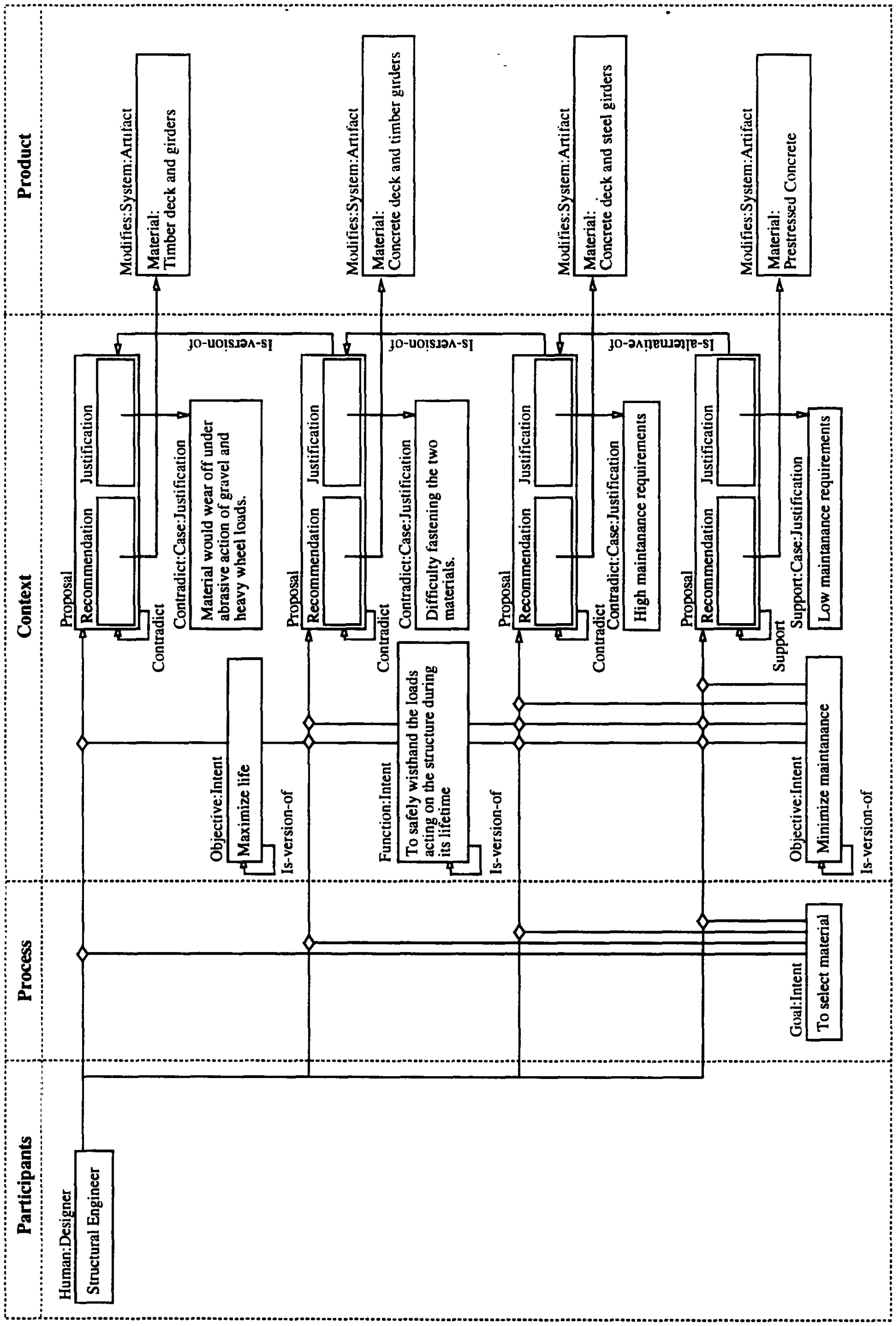
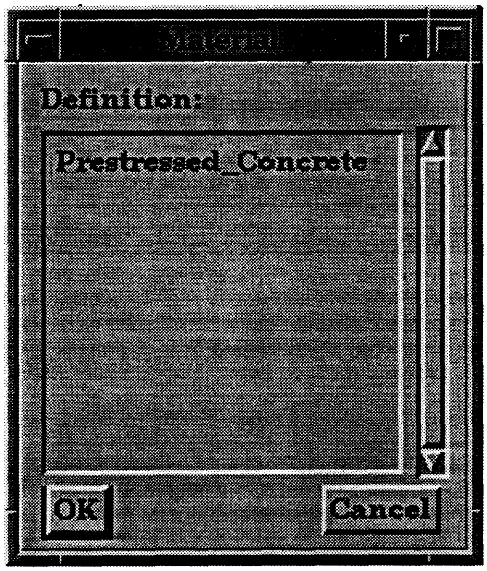


Figure 7-6: Proposals presentation for selecting the bridge material

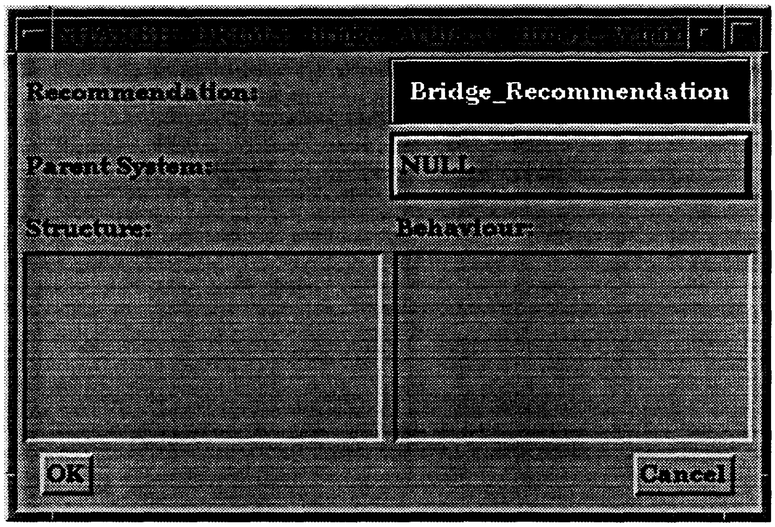


Material

Definition:

Prestressed_Concrete

Figure 7-7: Type of material in the bridge artifact



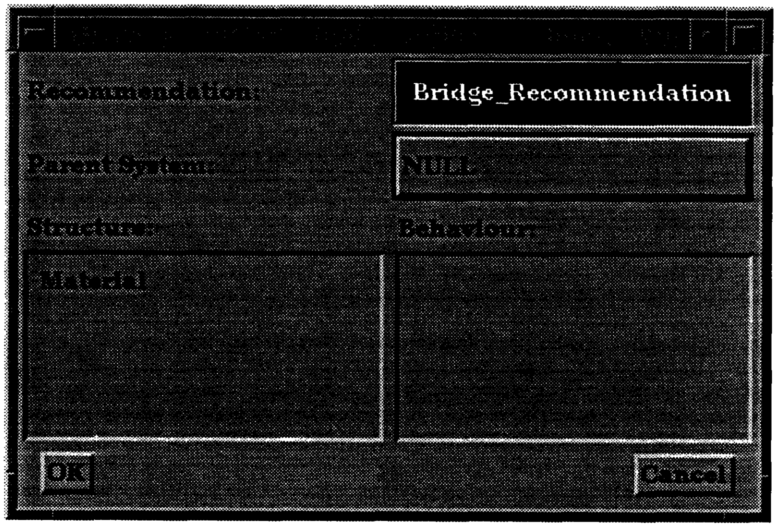
Recommendation: Bridge_Recommendation

Parent System: NULL

Structure:

Behaviour:

Figure 7-8: Bridge artifact as a place holder



Recommendation: Bridge_Recommendation

Parent System: NULL

Structure:

Material

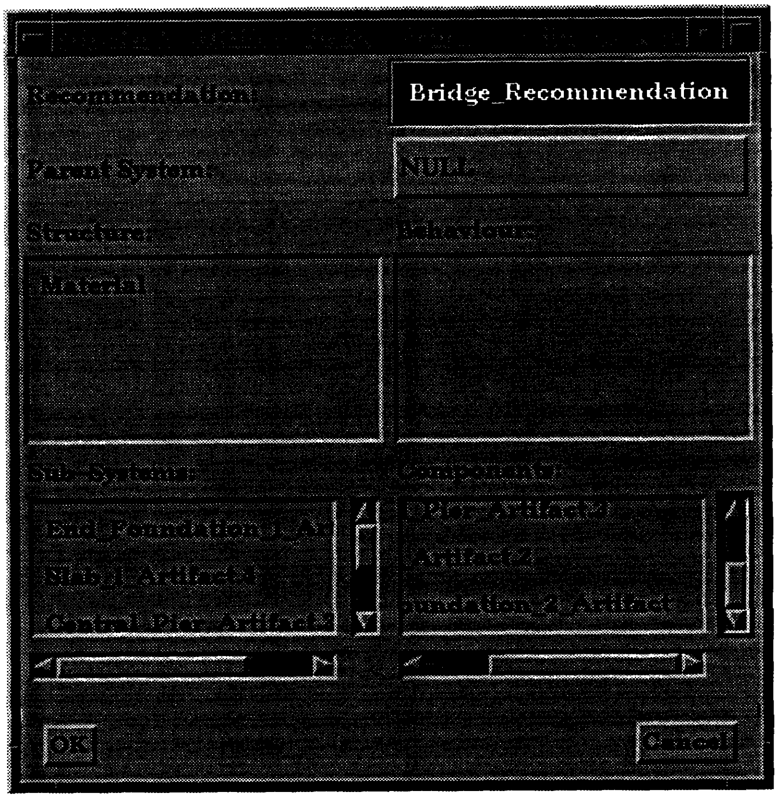
Behaviour:

Figure 7-9: Bridge artifact with a material in its structure

end-foundations, a slab, and a central pier at this stage. In addition, the system keeps the design intent hierarchy which led to this design. To design the bridge, the structural engineer selects the material and the form to withstand the load at minimum cost. These design intents relate to a series of proposals (see Figure 7-16).

SHARED-DRIMS oversees the various proposals made by the different designers and how they may conflict with each other. In this case, the structural engineer has proposed a prestressed concrete bridge with two spans, a central pier, and end foundations. The physical shape information is maintained by a geometric modeler as shown in Figure 7-17.

This information is now presented to the other designers. The other designers revise the design and provide feedback. The contractor has a problem with the prestressed concrete proposal (see Figure 7-18). The contractor recommends the use of prefabricated prestressed concrete since he lacks construction facilities and local labor. The structural engineer accepts the prefabricated prestressed concrete recommendation (see Figure 7-19).



Recommendation: Bridge_Recommendation

Parent System: NULL

Structure:

Material

Behaviour:

Sub-Systems:

End_Foundation_1_Ar
Slab_1_Artifact-1
Central_Pier_Artifact-2

Components:

Pier_Artifact-2
Artifact-2
oundation_2_Artifact

Figure 7-10: Bridge artifact with material, sub-systems and components in its structure

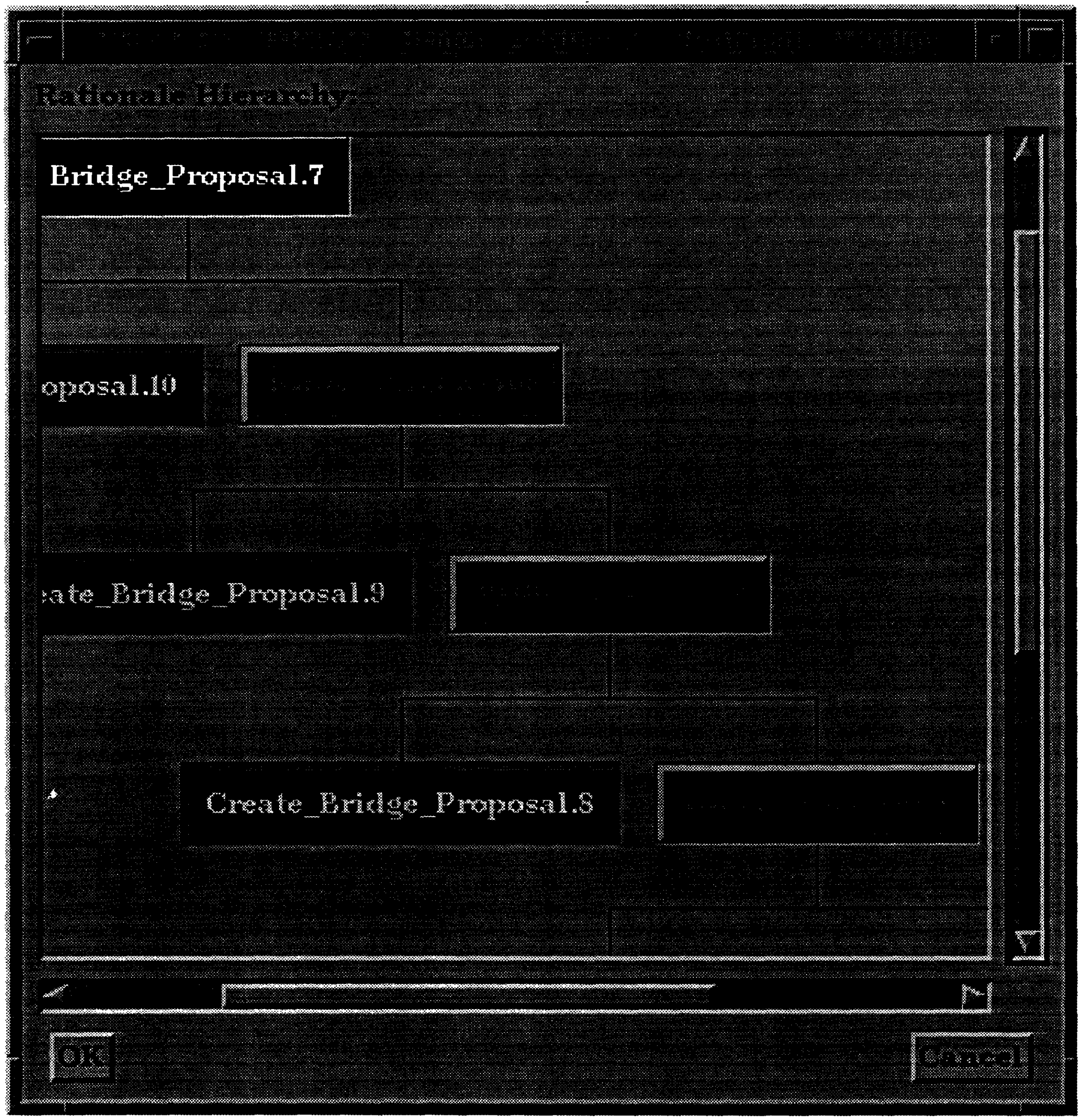


Figure 7-11: Version tree of the bridge artifact

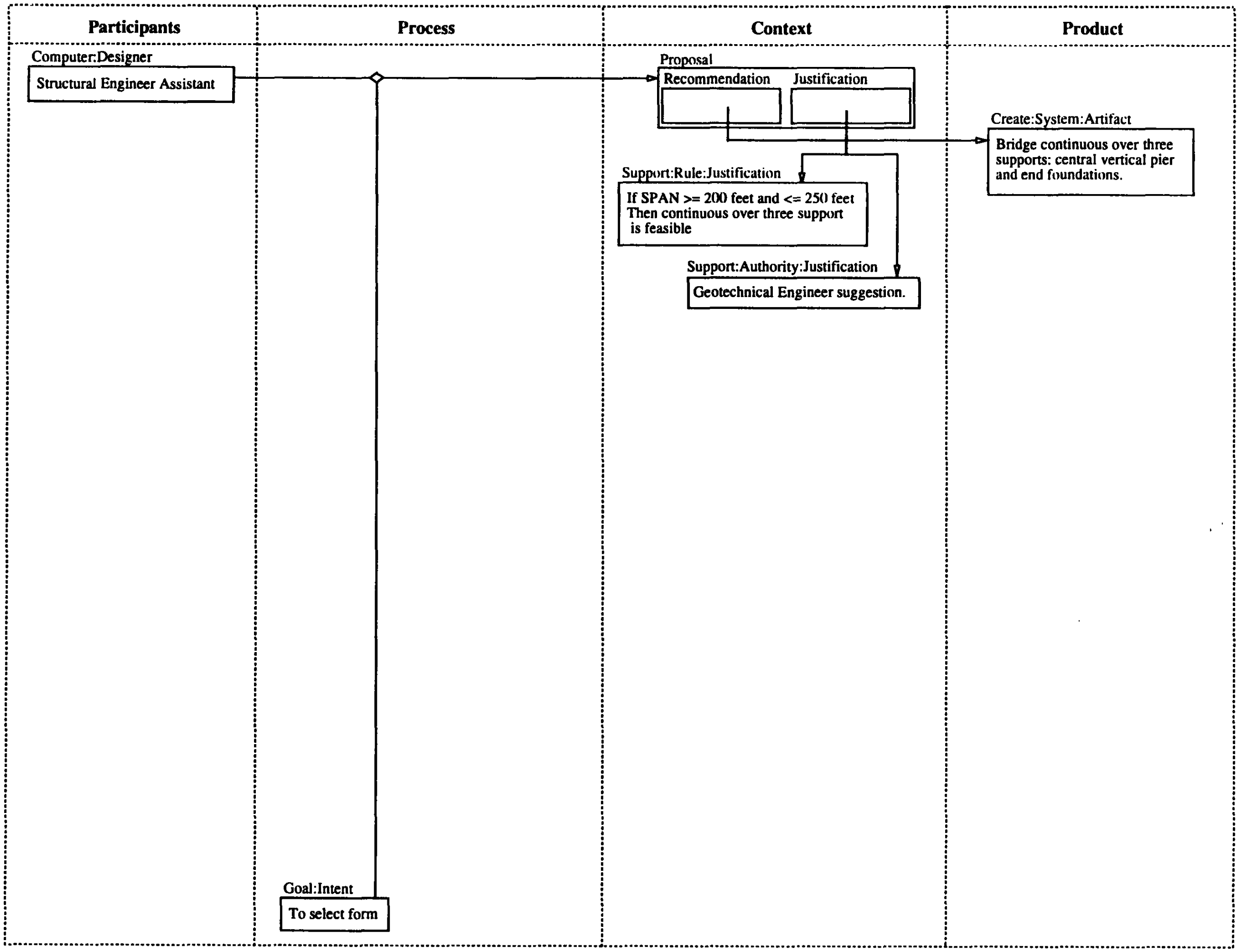
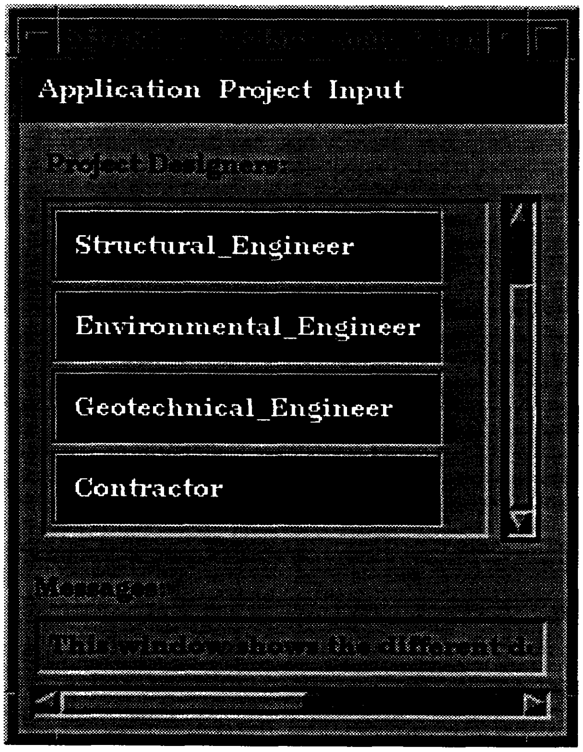

Participants
Process
Context
Product
Computer:Designer
Structural Engineer Assistant
Proposal
Recommendation
Justification
Create:System:Artifact
Bridge continuous over three supports: central vertical pier and end foundations.
Support:Rule:Justification
If SPAN >= 200 feet and <= 250 feet Then continuous over three support is feasible
Support:Authority:Justification
Geotechnical Engineer suggestion.
Goal:Intent
To select form

Figure 7-12: Initial proposal by SHARED-DRIMS to select the bridge structural form

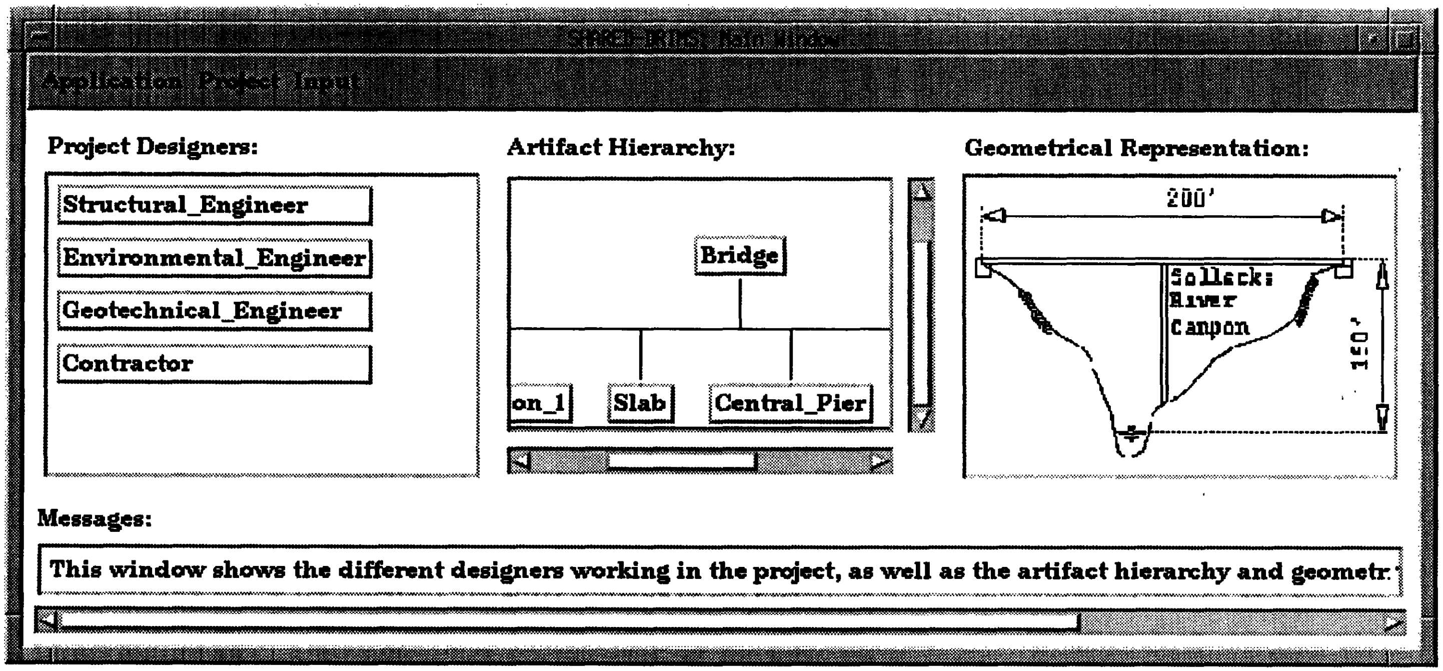
```
(RULE: Two_Span 20
IF
((CLASS: DRIM_S_Bridge OBJ: $x
(((length > 200.0)
AND (length == $dd ))
AND (largest_depth == $dd1))
) AND
(CLASS: DRIMIntent OBJ:  $a
((((((
((refer_to == "Goal") AND (refer_to == $re))
AND (action == "Select")) AND (action == $ac))
AND (object == "Form")) AND (object == $ob))
AND (modifier == "Bridge")) AND (modifier == $mo))))
THEN (
(EXECUTE OBJ: $a notify[Two_Span_Bridge_Selected])
(MAKE (CLASS:DRIMArtifact OBJ:End_Foundation_1
          (rule "Two_Span")
          (parent $x)
          (in $a)
          (ca $a)))
(MAKE (CLASS:DRIMArtifact OBJ:Slab_1
          (rule "Two_Span")
          (parent $x)
          (in $a)
          (ca $a)))
(MAKE (CLASS:DRIMArtifact OBJ:Central_Pier
          (rule "Two_Span")
          (parent $x)
          (in $a)
          (ca $a)))
(MAKE (CLASS:DRIMArtifact OBJ:Slab_2
          (rule "Two_Span")
          (parent $x)
          (in $a)
          (ca $a)))
(MAKE (CLASS:DRIMArtifact OBJ:End_Foundation_2
          (rule "Two_Span")
          (parent $x)
          (in $a)
          (ca $a)))
(PRINT "Creating the Two Span  Bridge\n")
)COMMENT: "")
```

Figure 7-13: Rule for developing a two span bridge



This window shows the different d

Figure 7-14: SHARED-DRIMS: Main Window with the project participants.



This window shows the different designers working in the project, as well as the artifact hierarchy and geometr

Figure 7-15: SHARED-DRIMS: Main Window with the project intents, the product hierarchy of the bridge, the proposal hierarchy of the designer, and a rough sketch of the design.

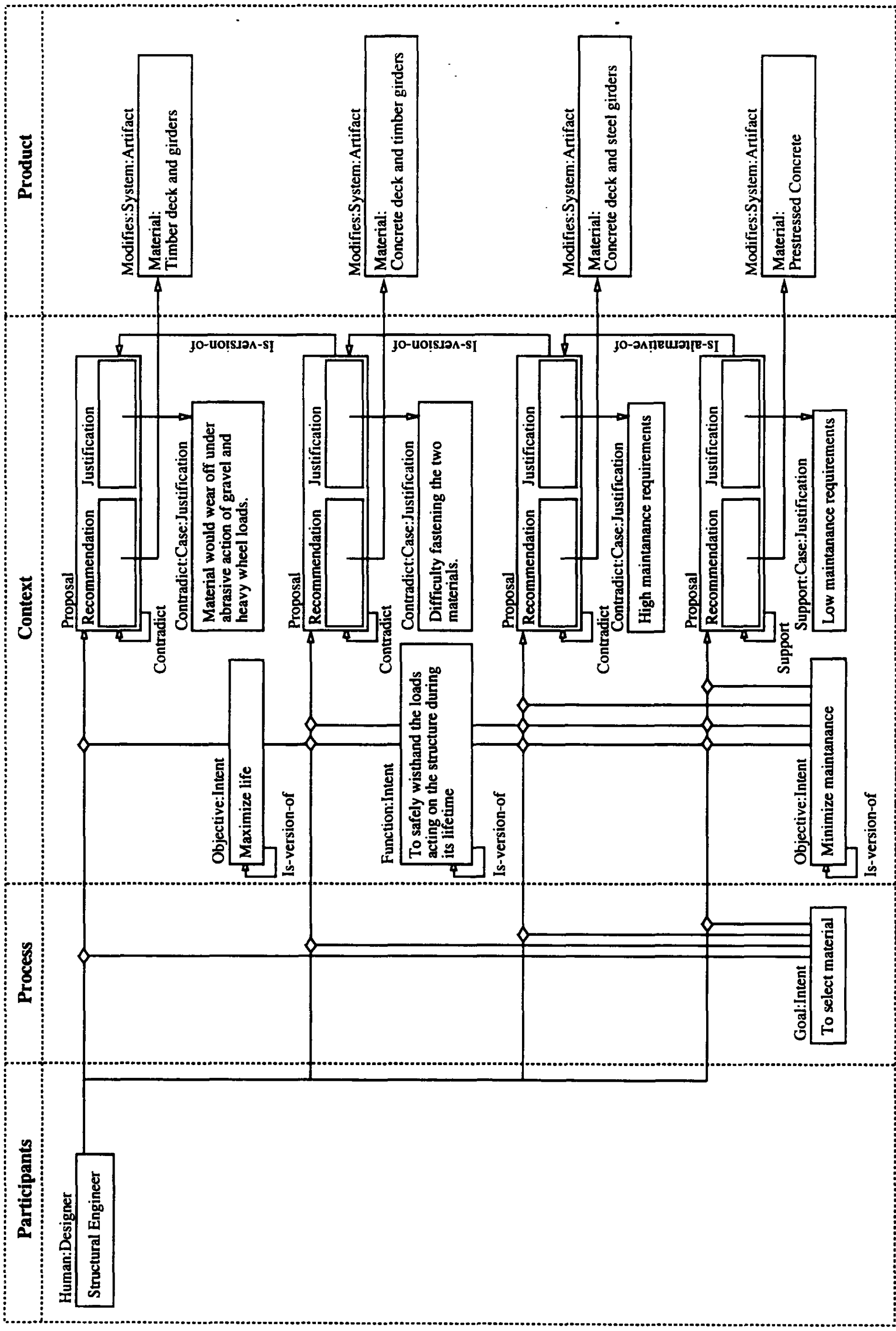


Figure 7-16: Bridge material design rationale

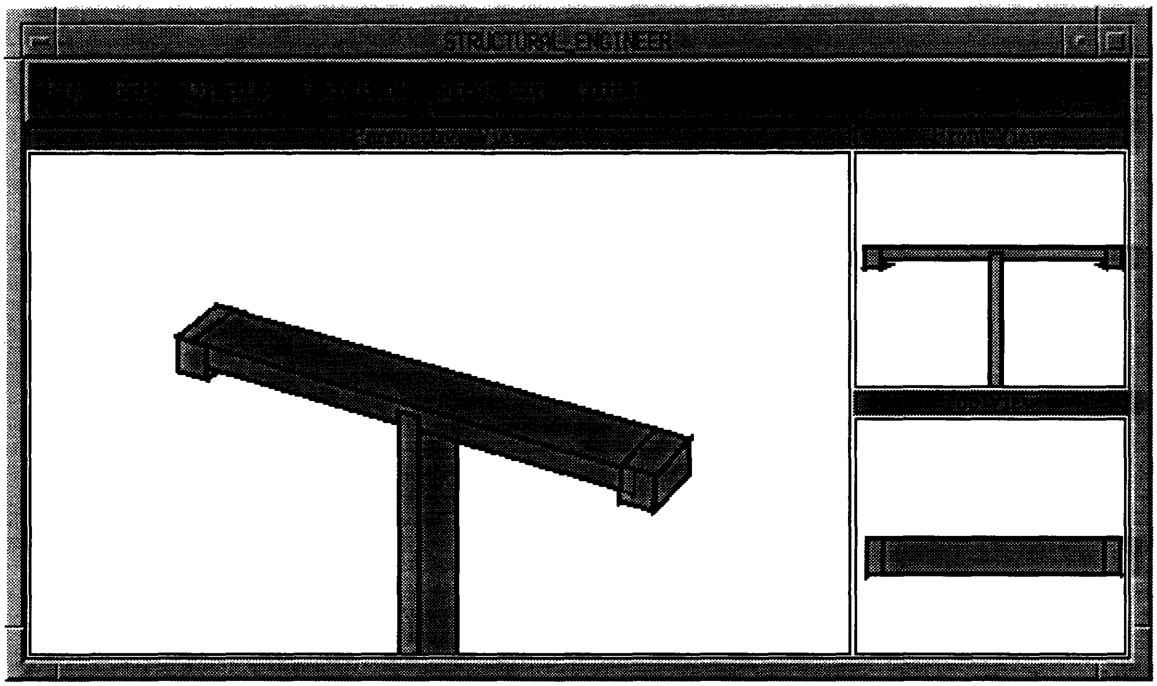


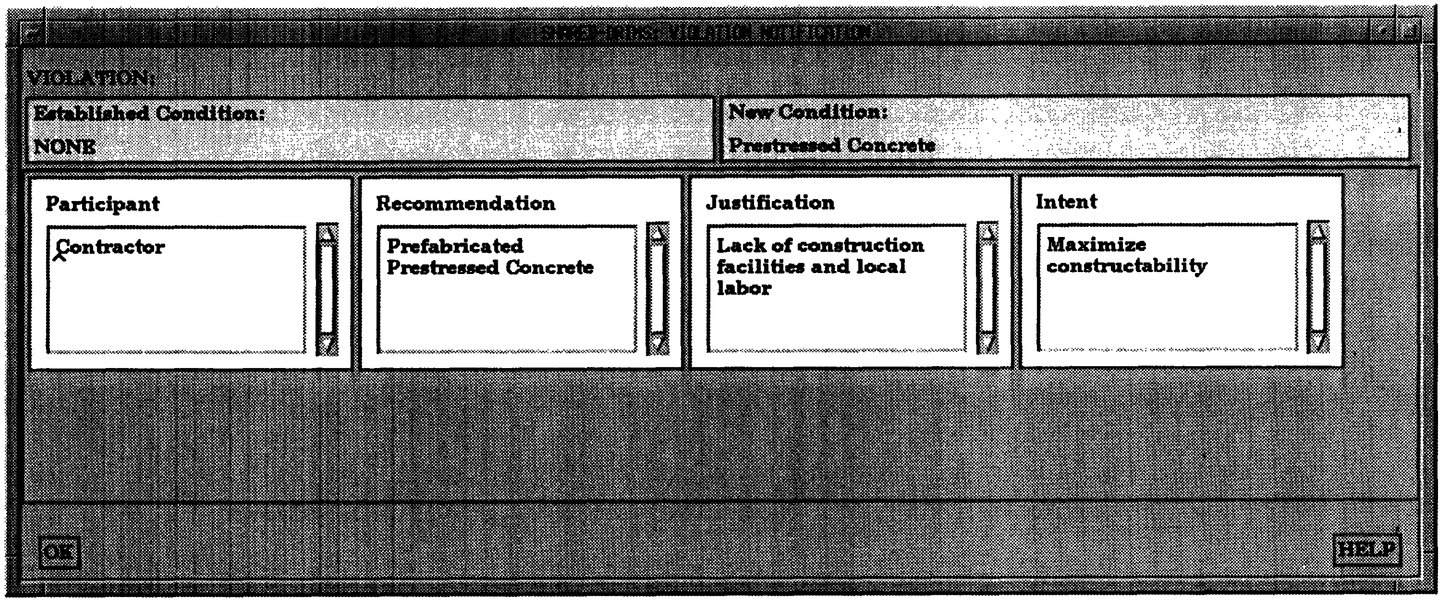
Figure 7-17: Display of the two span bridge in the non-manifold geometric modeler (GNOMES).

SHARED-DRIMS seeks confirmation that the change is due to conflicting proposal.

Similarly the proposal to use a central pier is rejected. SHARED-DRIMS informs that both the contractor and the environment engineer recommend the removal of the central pier (see Figure 7-20). The contractor cites lack of construction facilities as his justification and the environmental engineer is concerned that fish life may be affected by the construction. The structural engineer accepts the recommendation for the central pier removal. Now the changes are made to the design. These changes are reflected by the geometric modeler which shows the single span design (see Figure 7-21).

However, SHARED-DRIMS detects a new conflict due to this change. The single span design conflicts with the contractor's recommendation of using prefabricated prestressed concrete (see Figure 7-27). The justification presented is that only members of less than 78 foot length can be transported easily (see Figure 7-28, Figure 7-29, Figure 7-30, and Figure 7-31). Based on this proposal the structural engineer now revises the design to a three span bridge (see Figure 7-22). If the structural engineer did not understand the issues raised by the other participants, then, she can use a white shared-board, e-mail and video connections to communicate with the other participants (see Figure 7-23, Figure 7-24, Figure 7-25, and Figure 7-26).

SHARED-DRIMS detects another conflict with the geotechnical engineer's recommendation to place the supports near the base or the ends since soil exploration showed that the soil only has good support conditions to forces normal to the surface (see Figure 7-32). With this information, the structural engineer decides that she cannot change to two vertical piers. Based on design knowledge, SHARED-DRIMS and the structural engineer change the inclination of the piers to 60 degrees, that is, normal to the slope at the base of the piers.



SHARED-DRIMS: VIOLATION NOTIFICATION

VIOLATION:

Established Condition:
NONE

New Condition:
Prestressed Concrete

Participant
Contractor

Recommendation
Prefabricated Prestressed Concrete

Justification
Lack of construction facilities and local labor

Intent
Maximize constructability

Figure 7-18: SHARED-DRIMS: Violation Notification Window with the contractor rationale for rejecting prefabricated concrete as the material for the bridge.

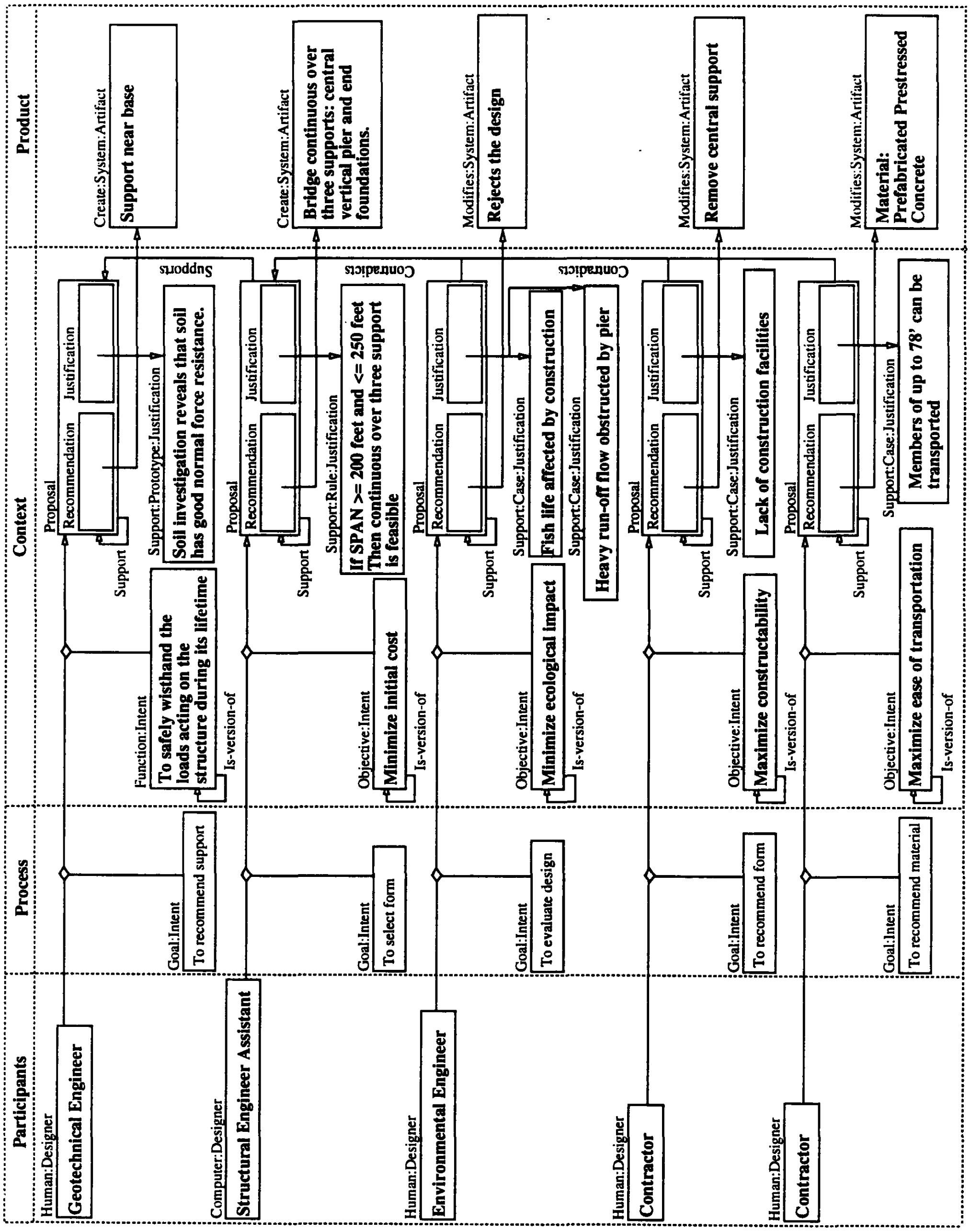
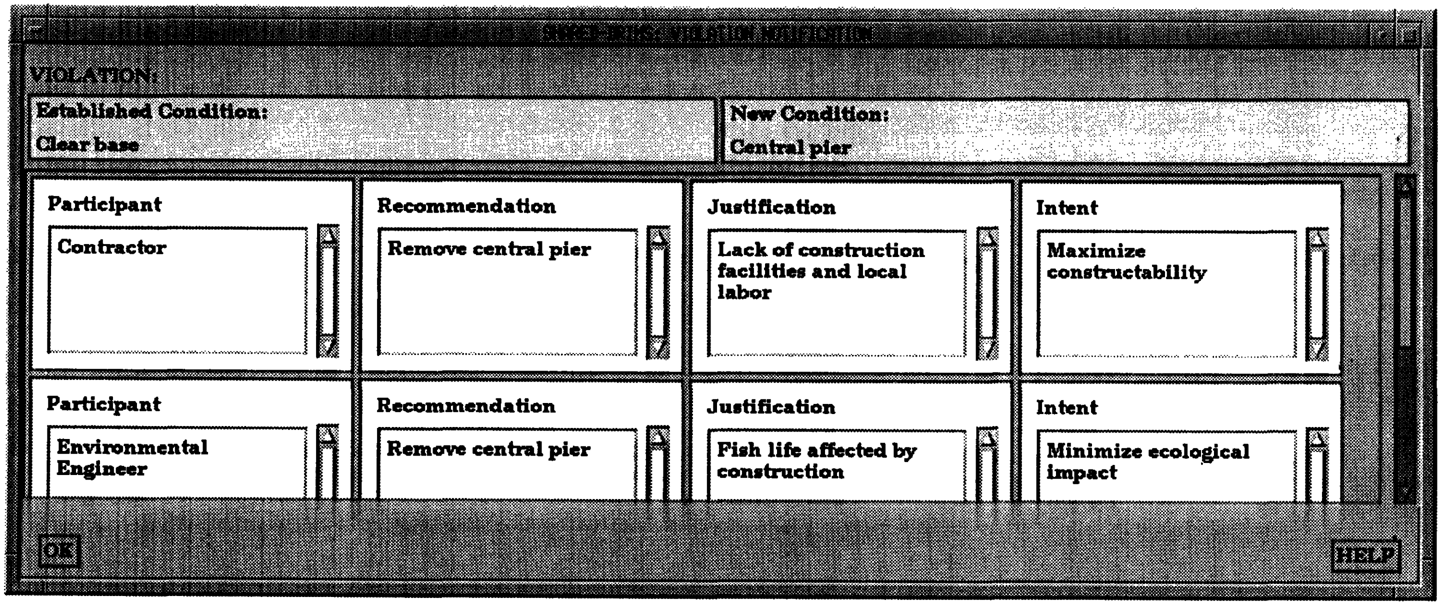


Figure 7-19: Material proposals presented by the different participants



VIOLATION:

Established Condition:
Clear base

New Condition:
Central pier

| Participant | Recommendation | Justification | Intent |
|---|---|---|---|
| Contractor | Remove central pier | Lack of construction facilities and local labor | Maximize constructability |
| Environmental Engineer | Remove central pier | Fish life affected by construction | Minimize ecological impact |

Figure 7-20: SHARED-DRIMS: Violation Notification Window with the rationale of the contractor and the environmental engineer which makes the usage of a central pier conflicting.

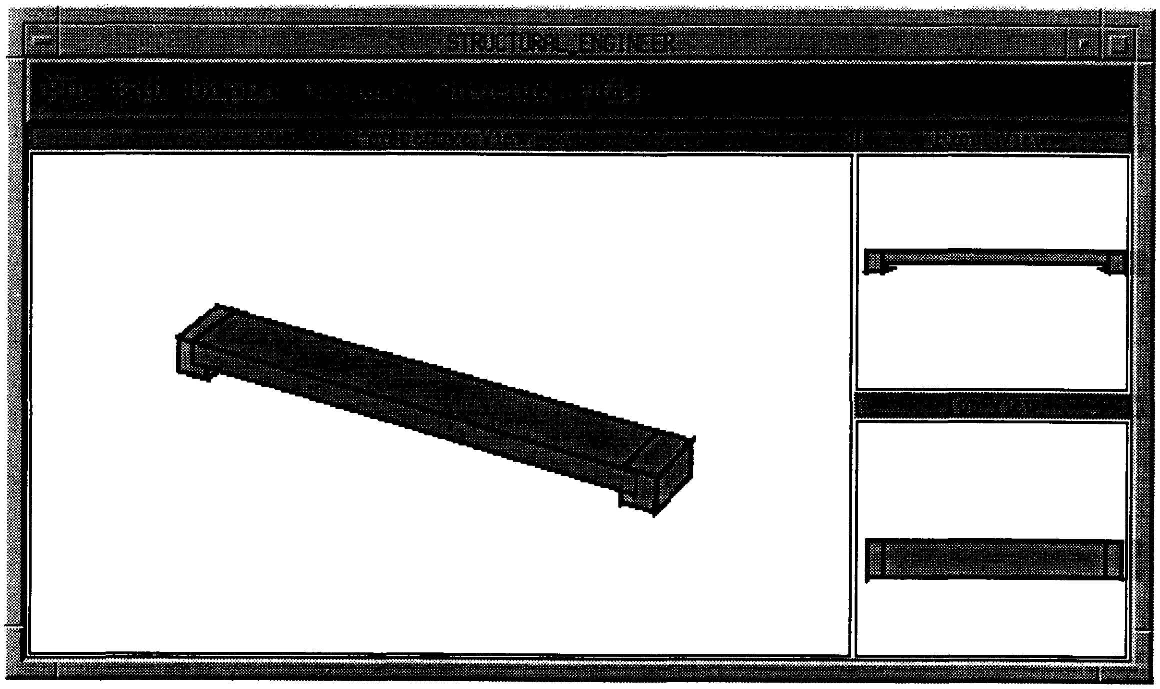


Figure 7-21: Display of the single span bridge in the non-manifold geometric modeler (GNOMES).

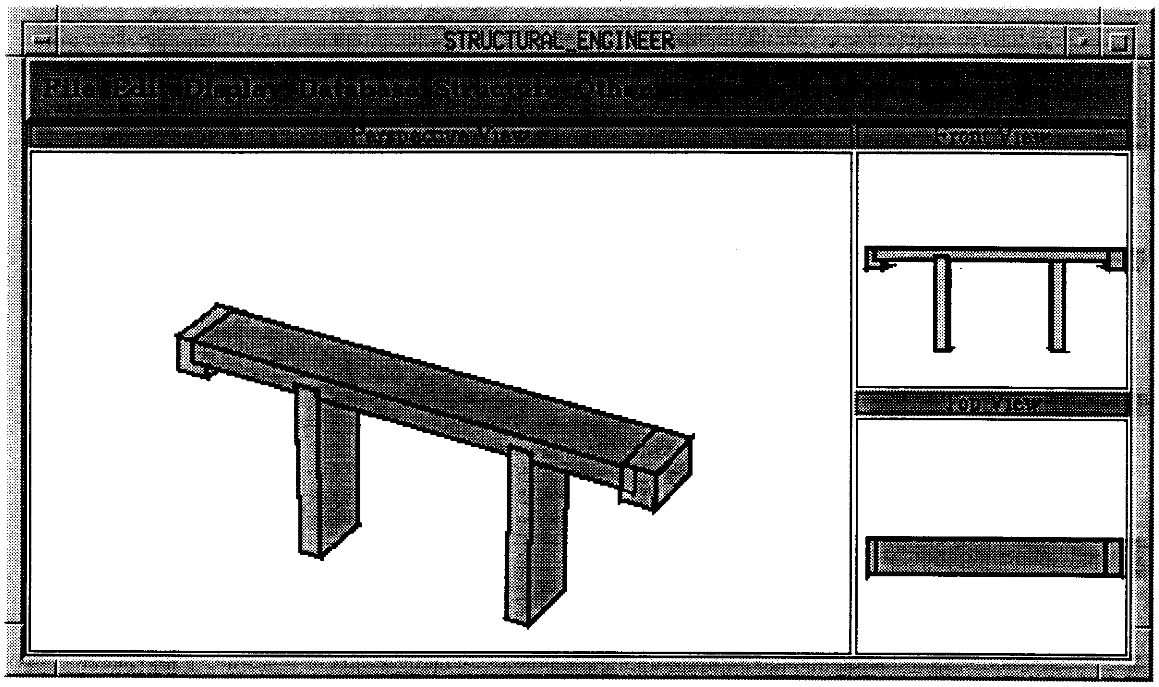


Figure 7-22: Display of the three span bridge with vertical piers in the non-manifold geometric modeler (GNOMES).

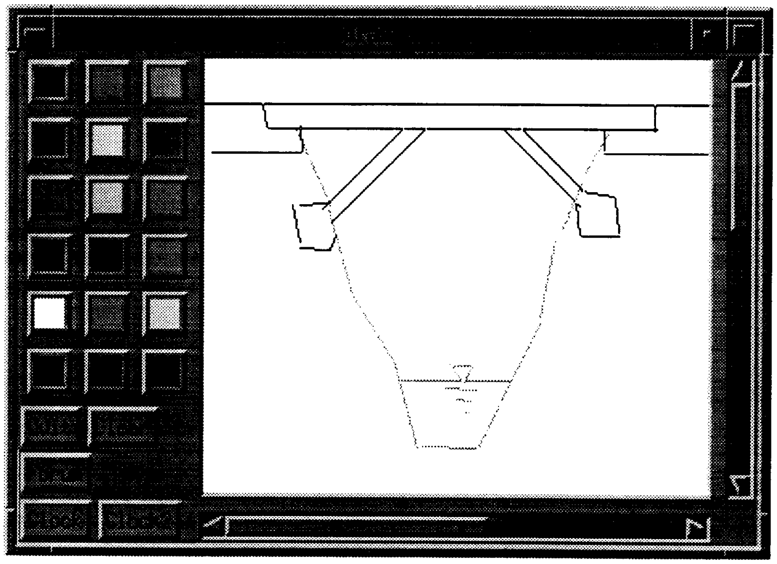
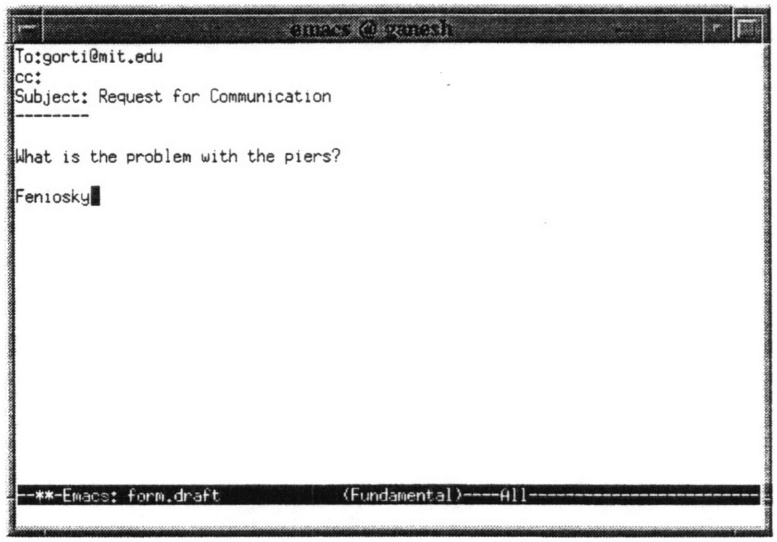


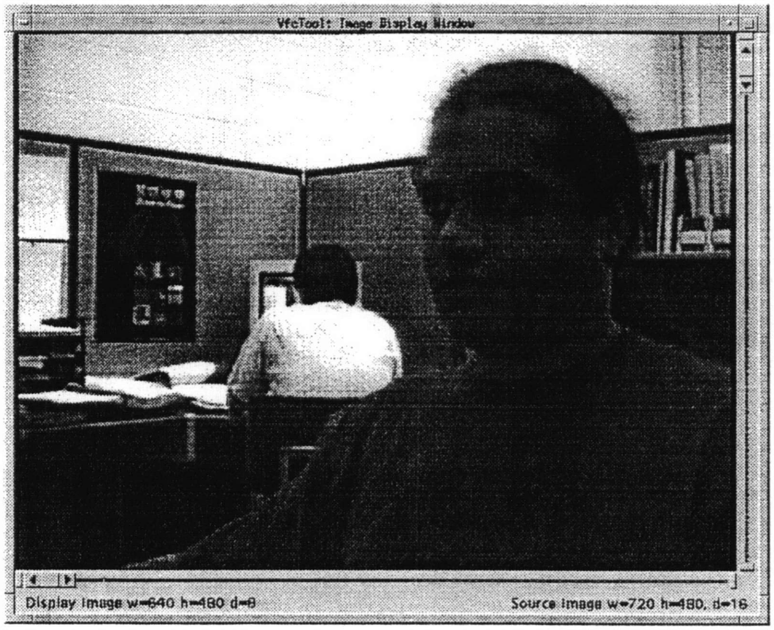
Figure 7-23: White Shared-Board for inter-machine communication



```
To:gorti@mit.edu
cc:
Subject: Request for Communication
--------

What is the problem with the piers?

Feniosky
```

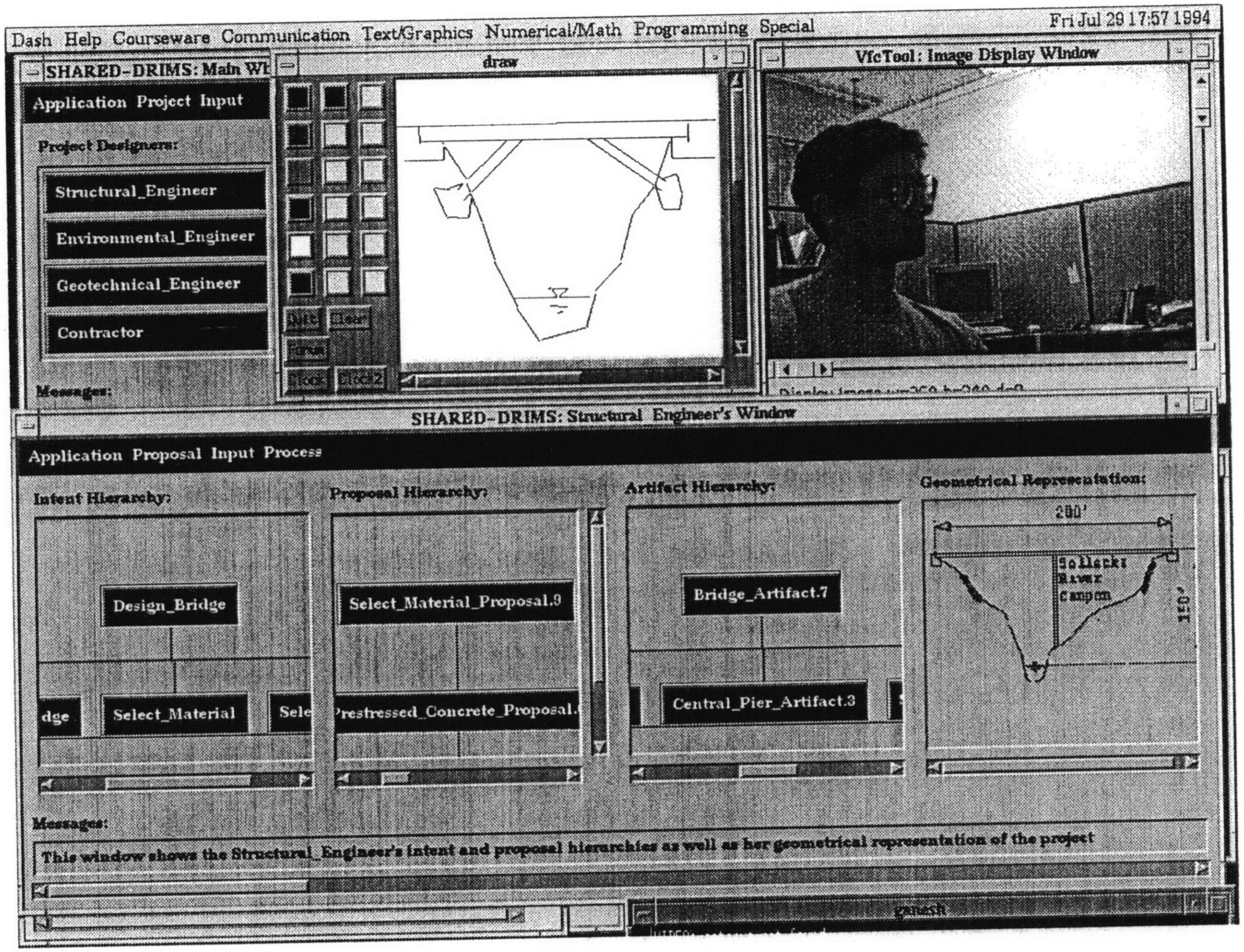
```
--**-Emacs: form.draft              (Fundamental)----All------------------------
```

Figure 7-24: E-mail window for sending messages to other participants



VfcTool: Image Display Window
Display Image w=640 h=480 d=8
Source Image w=720 h=480, d=16

Figure 7-25: Video window for visual inter-participants communication



This window shows the Structural_Engineer's intent and proposal hierarchies as well as her geometrical representation of the project

Figure 7-26: Complete screen with all the inter-participants communication tools

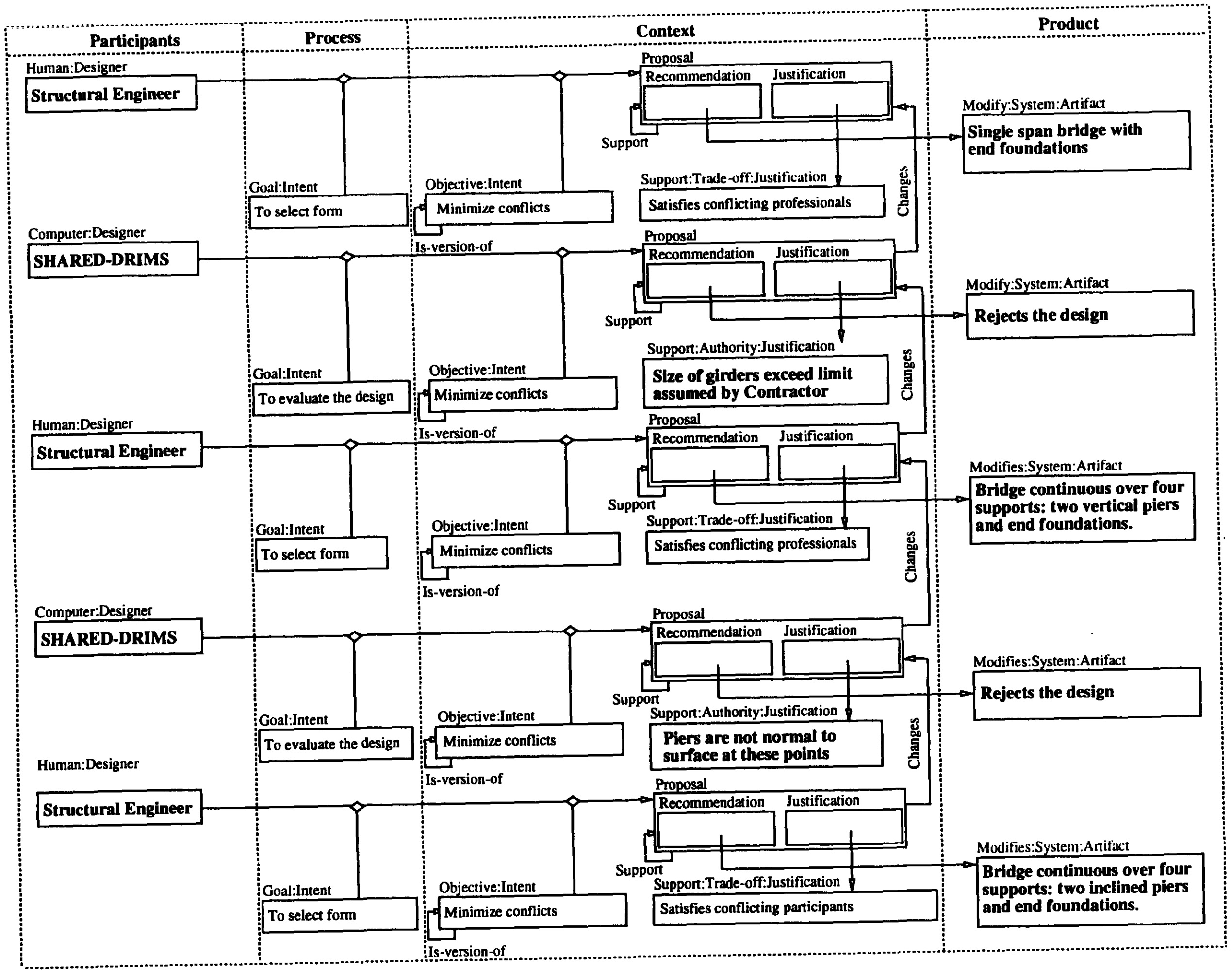


Figure 7-27: Form proposals presented by SHARED-DRIMS and the structural engineer

```
(RULE: MaximumLength 20
IF
(CLASS: DRIMSpec OBJ: $x
((attribute == "Length")
AND (value >= "78.0" ))
)
THEN (
(PRINT "Maximum Length exceded.\n")
(EXECUTE OBJ: $x notify[MaximumLength])
)
COMMENT: "")
```

Figure 7-28: Rule set by the contractor limiting the size of the members

```
DBoolean DRIMArtifact::put_value(char* slot,char* slot_value)
{
 if (strcmp(slot,"parent")==0)
  {
   COS_root *tempcr = (COS_root *)slot_value;
   tempcr = (COS_root *)os_workspace::current()->resolve(tempcr);
   insert_subs(tempcr);            /* Building the parent child connection */
   tempcr->set_parent_artifact(this);
   run_new_ie((COS_root*) tempcr);

   return 1;
  }

/* Other parts of this procedure has been deleted */

}
```

Figure 7-29: C++ code for linking artifacts

```
void run_new_ie(COS_root *obj_che)
{
/* Setting the Globals for versioning */

current_designer_configuration =
 (GNconfiguration *)Dmanagerbase::db->find_root("Sollecks_configuration")->get_value();

if(current_designer_configuration->frozen())
 {
   current_designer_configuration->new_version();
   current_designer_configuration =
    (GNconfiguration *)os_workspace::current()->
          resolve(current_designer_configuration);
 }

int temp = mode ; /* Since mode is a global variable, we must keep track
             of what we were doing before we came in here. */

Ie* temp_ie = theInferenceEngine ;

/* Here, we define the constraint and evaluate them */

DRIMConstraint *constrain =
 new (Dmanagerbase::db) DRIMConstraint(obj_che);

constrain->check_constraint();

mode = temp ; /* and restore the status */

/* Now loop throught the working memory of the things in the ie.
  Whatever working memory elements have been touched, will have to be
  recorded in the old engine. As a temporary measure, however, the old
  ie is simply being reset, so that all changes are processed again. We
  remove the copying from new to old due that the constraint are not
  touching any element. */

theInferenceEngine = temp_ie ;
}
```

Figure 7-30: C++ code for checking constraints

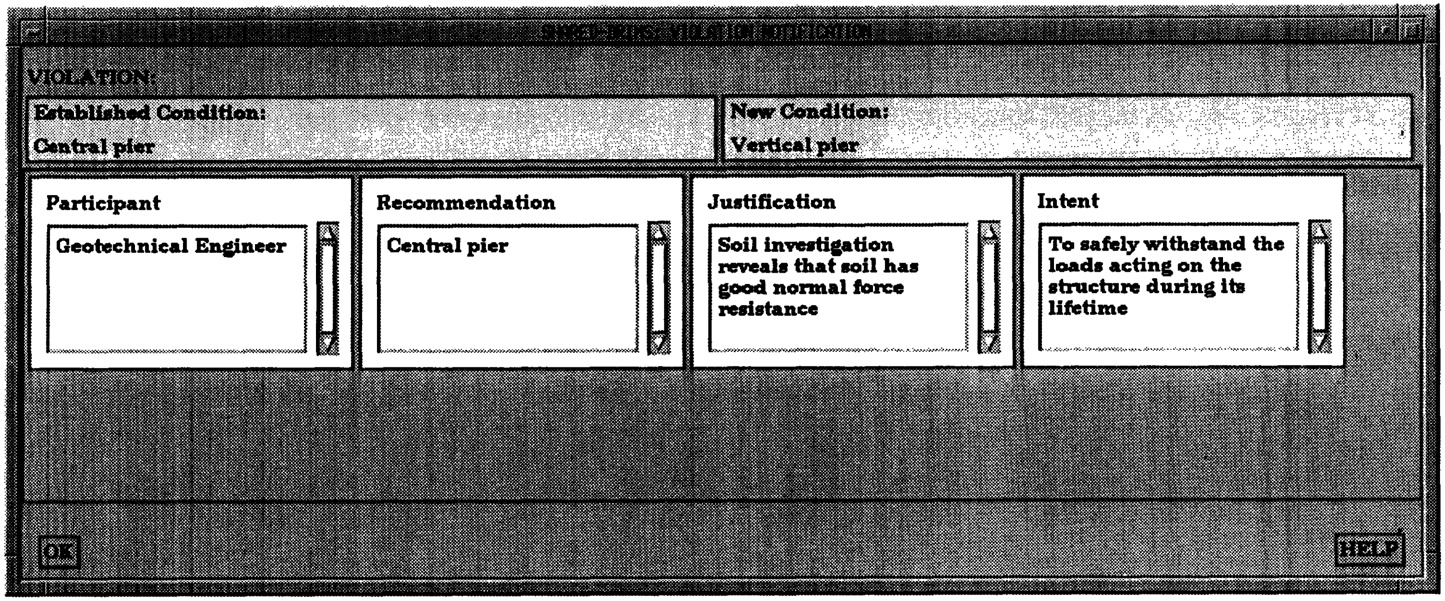
```
void DRIMConstraint::check_constraint()
{
 Ie ie;                    /* An inference engine object */

/* creation of a first instance of Forward—Chainer
   creates the associated RETE network representing a Rule file: */

 ie.reset() ;              /* Resetting the inference engine */
 ie.parse(rule_file);          /* Parsing the rule file */
 ie.load1_wme((COS_root*)recomm);   /* Loading the object */
 if(SD_trace)
  cout << "LOADED" << endl;
 ie.run();                 /* Running the inference engine */
 if(SD_trace)
  cout << "RUN DONE" << endl;
}
```

Figure 7-31: C++ code for running the inference engine

SHARED-DRIMS finds this design to be consistent and satisfactory to all participants (see Figure 7-33). Figure 7-34 shows the final design as presented by the geometric modeler.

## 7.3 Implementation

SHARED-DRIMS is implemented using $ObjectStore^{TM}$, C++ and X-Windows/Motif on a SUN-SPARC workstation. A hybrid representation for the leaf nodes of the model (e.g., Objective, Constraint, and Rule) is used, that combines structured data and free text. This dual representation allows for encapsulation of both human and computer rationale. In addition, the system is separated into two layers: the general layer and the specific domain layer. The general layer is domain independent and provides the constructs for representing design rationale. The specific domain layer has the knowledge of how to design and resolve



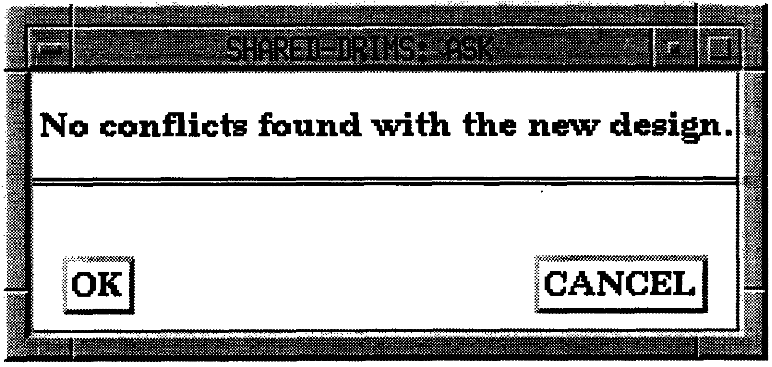
VIOLATION:

Established Condition:
Central pier

New Condition:
Vertical pier

| Participant | Recommendation | Justification | Intent |
|---|---|---|---|
| Geotechnical Engineer | Central pier | Soil investigation reveals that soil has good normal force resistance | To safely withstand the loads acting on the structure during its lifetime |

Figure 7-32: SHARED-DRIMS: Violation Notification Window with the rationale of the geotechnical engineer which makes the usage of a vertical piers conflicting.



No conflicts found with the new design.

Figure 7-33: SHARED-DRIMS: Ask Window which informs that the system could not find any conflict with the new design.

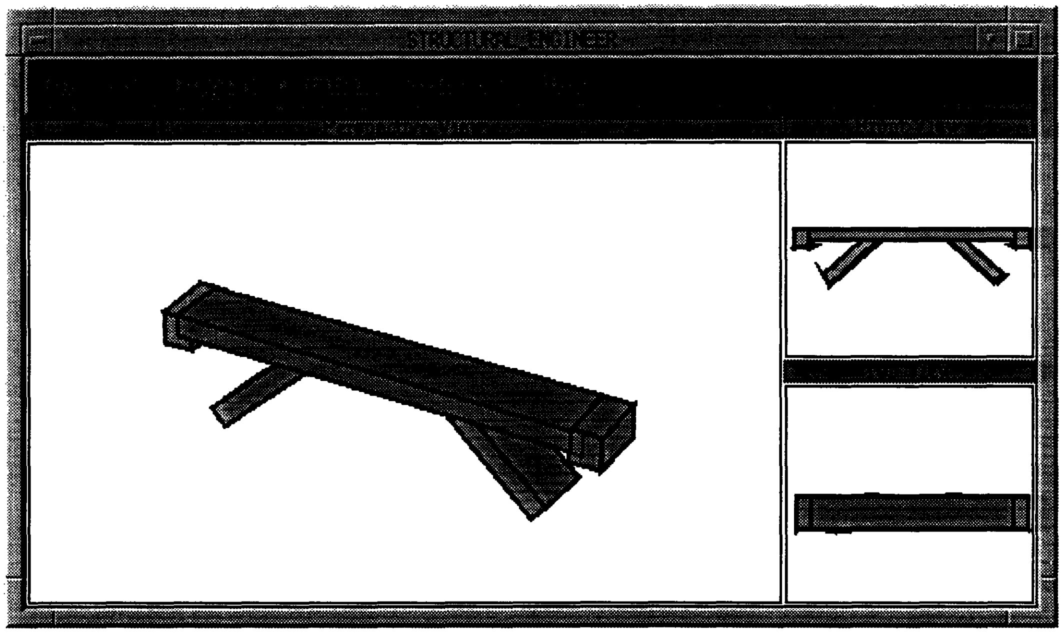


Figure 7-34: Display of the three span bridge with inclined piers in the non-manifold geometric modeler (GNOMES).

conflicts about a particular domain.

## 7.4 Chapter Conclusions

This example shows how SHARED-DRIMS supports conflict mitigation during the development of large scale engineering systems. SHARED-DRIMS is used by a structural engineer and three other professionals to reduce the number of their interactions to two from the five that the early process took. This early identification and resolution of the conflicts helps mitigate conflicts. As this example demonstrates, there were some deficiencies with the original process. First, the original process took a long time to deliver. Second, the original process might have produced poor quality products. In terms of delivery time, there were three main deficiencies. First, some information might have been lost, such as information regarding the reasons why a specific recommendation was made. Second, the participants would have to regenerate the information lost if they are asked for it. Third, the original process had interactions that could be avoided if information was available about why a design had been rejected or why a recommendation was made. For example, the selection of a single span bridge is not presented to the participants as an alternative, since the structural engineer know the reasons for the rejection of the long pier by the contractor. In terms of the quality, there was one main problem. Constraints set by participants through assumptions might have been violated. Violations might have gone undetected during the initial stages and might have been detected at a later stage. The changes at later stages would be made under heavy time and cost constraints. Designers would tend to focus on the problem area and disregarded the overall optimization of the product. All these problems are avoided with SHARED-DRIMS. The system only allows for two interactions between the different designers helping to mitigate conflict before they are spread over the whole design.

# Chapter 8

# Conclusions

*Government and co-operation are in all things the laws of life; anarchy and competition the laws of death.*

John Ruskin, *Unto this Last, Essay iii, 54, 1862*

## 8.1 Introduction

The use of computer-aided design, as well as the development of a computer-supported cooperative work environment, for the design process can produce fundamental changes not only in the accuracy and speed of the design, but also in the design process itself. However, there are still problems with the vast amount of data produced by the design process. Certain kinds of vital information – usually implicit on the process, often having to do with *why* certain actions are taken – are usually lost in large projects. One reason may be that these types of information, while important, are never expressed explicitly to be readily captured and retrieved. Usually the information that is related to the relationship between the parts of an artifact and the context in which they are designed to work is lost. Thus, data inconsistencies are difficult to detect and if they are detected, it is difficult to access the causes of such inconsistencies. To mitigate conflicts a system that can deal with the previous problems is needed. Thus, the key technical problems that such a system faces are: 1) determining where a problem lays, and 2) determining why a problem occurs.

To solve these data capturing and manipulation problems, this dissertation has developed a model –DRIM– that represents the design rationale of an artifact. Once the design rationale has been captured, a conflict mitigation system –SHARED-DRIMS– detects inconsistencies in the data and pinpoints the basic reasons for the occurrence of those inconsistencies.

The DRIM model captures the design rationale through the use of the following five constructs: designer, proposal, recommendation, intent and justification. Each one of these constructs can be instantiated on objects that may be related through composition, version and/or abstraction trees. This model also provides relationships that ensure a structure is followed during the process. However, this structure does not prevent the iterative and collaborative nature of the process to be present; on the contrary, it supports it.

The DRIM model and the SHARED-DRIMS system presented in this work are implemented in an object-oriented style: the designer, proposal, recommendation, intent, and justifications constructs are independent objects which contain methods that specify their role and behavior in any context.

The SHARED-DRIMS system performs five functions: rationale capture, conflict detection, conflict causes, conflict resolutions and conflict prevention. The rationale capture function uses domain knowledge, past designs and interaction with the designers to capture design rationale. The conflict detection function uses physical constraints to detect inconsistencies. The conflict causes function retrieves rationale information and maps it into the relevant context. The conflict resolution function uses domain knowledge, past designs and interaction with the designers to provide recommendations that solve the conflict. Finally, the conflict prevention function uses domain knowledge, past designs and interaction with the designers to access the effect of decisions made during the conflict resolution.

The use of the model and the system presented in this dissertation improves the design process, both in terms of time and quality. Design decisions do not need to be rehashed several times during the process simply because no one can recall how the decisions were reached previously, which means saving time. In addition, the ability of the system to tell the designers very early in the process various problems with their assumptions or data allows them to correct the problems early and avoid repeating the same errors on other designs.

## 8.2 Benefits of the Model

In order for a model like the one proposed in this work to be successful, some considerations for the user should be given. The users need to know the palpable benefits that such a model could provide. Among these benefits is the characteristic that this model is an improvement over the existing design process. Designers have to provide the assumptions they make during the design, as well as the specifications of the design artifact. However, the artifact documentation is created after the design is made. Therefore, assumptions and decisions made early in the design process may be lost and cannot be provided. This model facilitates the design documentation to be created as the design proceeds. Thus, assumptions and decisions are recorded at the time they are made and therefore will not be lost. In addition, the model uses a mixture of structured and semi-structured representations for capturing the design rationale. Hence, the users will not be exposed to radical changes in their work technologies. This model is structured in that the objects could be defined using object-attribute-value representation so to allow computer inferencing. This model is semi-structured in the sense that there is a structured relationship between the elements that compose the design rationale, but the user comments and justifications can be expressed in natural language, digital photographs, digital video or a combination of these.

The users of the model may have the benefit of knowing not only what was decided but also what was discussed in physical or electronic meetings in which they were not involved. The organization in which the model is used can benefit by creating personnel independence. For example, if a designer who played an important role during the preliminary design leaves the company before the detailed design phase is completed, the detailed design need not suffer from information loss. Because the early design rationale was captured using the model, and made available to all the agents involved in the project shortly after it was recorded, much of the logic behind the design will have been retained. This is in contrast with current practice where the designers working in the detailed design phase would not have been able to answer questions related to *why* some decisions were made if a key designer were not available.

Another benefit of using the model for capturing design rationale is that it includes all the justifications for the decisions made in the design process. In addition the model includes the context in which the justifications were made (i.e. relationships between design problems). It is easier to determine *why* a decision was made without the need for actually replaying the discussion generated for decision-making. For example, a construction consultant may recommend a structural slurry wall for an earth retention wall system. After a certain amount of time, some questions may be asked about appropriateness of the decision for using a structural slurry wall. If the design rationale were not recorded, the designers would need to ask the individuals involved in that decision. If the individuals were not available, the designers would need to re-generate the decision process. However, if the model is used to record the design rationale, the designers need only ask the system to present all the elements involved in the decision of the structural slurry wall. The system will re-play that the structural slurry wall was selected for lowering the risk for disturbing adjacent buildings and that this goal was very important. With this information and more, the designers may re-evaluate the decision using all the arguments made during the initial design, as well as the arguments that the questions generated. The access to the early decision-making process improves the designer's confidence that all dependencies from early choices are taken into consideration.

This model may help detect errors during feasibility analysis, problem identification and formulation, design, construction, and use of an artifact. It can also influence effectiveness of project team meetings, inter-organizational communication, and project management in general.

- **Feasibility Analysis and Problem Identification and Formulation**
  During these phases, the model helps the individuals working on the project to quickly understand the problem that they are trying to solve. Explicitly stating the issues or problems to be addressed provides a framework not only for the discussion about the problem, but also for the entire development of the project. By presenting the need identification, evaluation and justification in terms of proposals, recommendations, intents, and justifications, weak supporting arguments are apparent and assumptions are made common knowledge.

- **Design, Construction and Consumption**

  A model such as the one proposed helps the project team detect design parts that may be omitted in both high and low level design reviews. In addition, the conflict mitigation system can detect early data inconsistencies and find the reasons for these changes. In other words, design errors are detected early in the process when they are less costly to repair.

- **Project Team Meetings**

  Sometimes in meetings, discussions tend to digress from the intended topic. By recording the discussions, open issues raised during tangential discussion could be saved to be addressed later. Returning to the original topic could be done quickly by reviewing what had been recorded prior to the tangential discussion. Another benefit is the use of SHARED-DRIMS for setting meeting agendas: attendees are provided with a set of design/decision problems and, whether the topic had been discussed before. They are also presented with any innovative alternatives and arguments. The agenda preparation by a system based on this model is useful when the design/decision problems being discussed are interrelated in ways that are not obvious.

- **Inter-organizational Communication**

  Conversations among different organizations carry the risk of the different priorities between organizations. The difference in priorities can cause some conversations to be forgotten or misunderstood. Thus, a lot of time is spent in reconstructing previous conversations. Another problem is when new personnel is assigned to a project, a record of past discussions and the process by which decisions were handled is needed to bring new people up to the required level of understanding quickly and easily.

- **Project Management**

  During a project, some design/decision problems are promptly resolved, but other remains open indefinitely. The model helps tracking these open design/decision problems. With the use of the model, the individuals involved can have all the previously information for these open problems and can make decisions without additional delay. In addition, the use of SHARED-DRIMS reduces the analysis burden on project managers by highlighting problem areas and the causes of these problems.

## 8.3 Contributions

In summarizing this work, the contributions can be divided in two parts. First, a model for representing design rationale was developed. This model provides primitives for representing design knowledge in terms of the reasoning process used by designers in generating an artifact satisfying their design intents. This model also takes into consideration the different collaborating participants and is used to provide active computer support for capturing designers' reasoning process. The model allows human designers interacting with computers to record their design rationale.

The second contribution is the computer support for conflict mitigation. SHARED-DRIMS can automatically resolve known conflicts when solutions are available in its knowledge base. In addition, the system provides hypotheses about the reasons for conflicts during the conflict resolution process. These hypotheses, once verified by the designer, can be used for better coordination and negotiation. This, in turn, will enhance the communication during the design process and consequently increase the productivity of the AEC and other related industries.

## 8.4 Future Research

SHARED-DRIMS successfully addressed the issues raised in the example of Section 2.2.5. However, many other issues need to be resolved for the effective implementation of SHARED-DRIMS in a real world working environment. These issues can be summarized in eight parts:

1. Testing of both the model and the system have been limited to small-scale problems dealing with conceptual design. Further testing is necessary to validate the extensibility of the model and its applicability throughout the life-cycle of large scale design-construction problems. The system has to be tested in a normal working environment where there are time pressures and organizational constraints. Exploration of how the designers use the system and cope with its limitations is needed.

2. There is a need to determine what other means could be used to capture reasoning information without disruption of designers' normal work style. Designers have expressed interest in using the system, but are concerned about spending time inputting the rationale. In the current system, there is knowledge incorporated in SHARED-DRIMS which helps the designer to record the rationale. However, the knowledge needs to be recorded by a knowledge engineer. Thus, techniques that allow easy input of design rationale need to be explored.

3. There is the question of the ratio between the cost to the designers and the organizations to record this information versus the benefit they will receive. Questions to be answered are: How much investment is the organization or the individual willing to commit to record this additional rationale? What is this individual or organization obtaining in return?

4. The organization and the system will change. The system will change how the organizations do business and the system will adapt to the organization. Information about this synergy is needed to maximize the usage of the system.

5. There is the question of how this information could be used by project control systems for reducing the impact of change orders on the cost and the schedule of the project.

6. There is a need to incorporate a set of robust general conflict mitigation strategies, as well as domain dependent conflict mitigation. As the system stands now, rules and primitive cases are used to store domain dependent conflict mitigation strategies. In terms of a general conflict mitigation strategy, the system provides the designers with conflict information and design rationale. However, it does not provide robust support. Work needs to be done to strengthen these two areas of computer supported conflict mitigation.

7. Social scientists have conducted various research experiments on the effect of human biases to negotiation. These research efforts try to explore the relationship of human background, knowledge and attitudes with the outcome of the negotiation. However, there has been little documentation of the effect of computer programs and methodology on the negotiation. Computer programs and methodologies have biases that are incorporated implicitly into these programs and models by the developers. One area of future research will be to explore the effect of computer programs and methodologies on the negotiation.

8. The classification of intent and justifications were found to be relevant in the case studies performed in this research. Some preliminary structures and semantic meaning were attached to these classifications. However, these meanings and

structures have not been completely tested and validated. Research is needed in studying the effect of the classifications on efficiently mitigating conflict.

# Bibliography

[Ahmed et al., 1992] Ahmed, S., Wong, A., Sriram, D., and Logcher, R. (1992). Object-Oriented Database Management Systems for Engineering: A Comparison. *Journal of Object-Oriented Programming.*

[Anandalingam and Apprey, 1992] Anandalingam, G. and Apprey, V. (1992). Multi-Level Programming and Conflict Resolution in International River Management. Mimeo, Center for Research in Conflict and Negotiation, Penn State University.

[Andrews, 1993] Andrews, J. D. (1993). The Design of a Rotating Annular Flume to Study the Erosion of Cohesive Sediments. Master's thesis, Massachusetts Institute of Technology.

[Anson and Jelassi, 1990] Anson, R. and Jelassi, M. (1990). A Development Framework for Computer-Supported Conflict Resolution. *European Journal of Operational Research*, 46(4):181–199.

[Banares-Alcantara, 1991] Banares-Alcantara, R. (1991). Representing the Engineering Design Process: two hypothesis. In Gero, J., editor, *Artificial Intelligence in Design '91*, pages 3–22. Butterworth-Heinemann, Oxford, England.

[Banerjee et al., 1987] Banerjee, J., W., K., and Korth, H. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Fransisco.

[Becker, 1989] Becker, J. (1989). Construction Technology and the Building Process. Foundation Decison Making Monarch Place - A Case Study (Class Assignment).

[Bradley and Agogino, 1991] Bradley, S. and Agogino, A. (1991). Design Capture and Information Management for Concurrent Design. *International Journal of Systems Automation: Research and Application*, 1(2):117–141.

[Brown, 1985] Brown (1985). Failure Handling in a Design Expert System. *Computer-Aided Design*, 17(9).

[Carey et al., 1989] Carey, M., DeWitt, D., Richardson, J., and Shekita, E. (1989). Storage Management for Objects in Exodus. In *Object-Oriented Concepts, Databases, and Applications, Kim, E. and Lochovsky, F. H. (Editors)*. ACM Press and Addison-Welsey.

[Casais et al., 1992] Casais, E., Ranft, M., Schiefer, B., Theobald, D., and Zimmer, W. (1992). OBST - An Overview. Technical report, Forschungszentrum Infromatik (FZI).

[Casotto et al., 1990] Casotto, A., Newton, A., and Sangiovanni-Vincentelli, A. (1990). Design Management based on Design Traces. In *27th ACM/IEEE Design Automation Conference*, pages 136– 141, Orlando, FA. IEEE.

[Chen, 1976] Chen, P. (1976). The Entity Relationship Model - Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1(1).

[City of New York Bureau of Water Pollution Control, 1980] City of New York Bureau of Water Pollution Control (1980). *Sewage Treatment Plant Design Review Handbook*. City of New York Department of Environmental Protection.

[Company, 1990] Company, A. (1990). *ABAQUS Manual.*

[Conklin and Begeman, 1988] Conklin, J. and Begeman, M. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331.

[Culbert, 1992] Culbert, J. (1992). Distributed Object Transport Streams ($DOTStreams^{tm}$): A Tool for Building Distributed Object Oriented Systems. IESL Technical Report IESL-92, MIT.

[Elmore et al., 1989] Elmore, P., Shaw, G., and Zdonik, S. (1989). The ENCORE Object-Oriented Data Model. Technical report, Brown University.

[Favela, 1993] Favela, J. (1993). *Organizational Memory Management for Large-Scale System Development.* PhD thesis, Massachusetts Institute of Technology.

[Favela et al., 1993] Favela, J., Wong, A., and Chakravarthy, A. (1993). Supporting Collaborative Engineering Design. *Engineering with Computers.* To appear.

[Firebaugh, 1989] Firebaugh, M. (1989). *Artificial Intelligence: A Knowledge-Based Approach.* PWS-Kent Publishing Co., Boston, MA.

[Fischer et al., 1989] Fischer, G., McCall, R., and Morch, A. (1989). Design Environments for Constructive and Argumentative Design. In *Proceedings of CHI'89*, pages 269–276, Austin, TX.

[Fraser and Hipel, 1988] Fraser, N. and Hipel, K. (1988). Using the Decision Maker Computer Program for Analyzing Environmental Conflicts. *Journal of Environmental Management*, 27:213–228.

[Fromont and Sriram, 1992] Fromont, B. and Sriram, D. (1992). Constraint Satisfaction as a Planning Process. In Gero, J., editor, *Artificial Intelligence in Design '92*. Kluwer Academic Publishers, London, England.

[Ganeshan et al., 1991] Ganeshan, R., Finger, S., and Garrett, J. (1991). Representing and Reasoning with Design Intent. In Gero, J., editor, *Artificial Intelligence in Design '91*, pages 723–736. Butterworth-Heinemann, Oxford, England.

[Gantz, 1989] Gantz, J. (1989). Sizing Up the AEC Market. *Computer Graphics World*, pages 43–45.

[Garcia and Howard, 1992] Garcia, A. and Howard, H. (1992). Acquiring Design Knowledge Through Design Decision Justification. *AI EDAM*, 6(1):59–71.

[Gorti and Sriram, 1993] Gorti, S, R. and Sriram, D. (1993). CONGEN: An Integrated Approach to Conceptual Design. *International Journal of CAD/CAM and Computer Graphics*, 8(2):135–150. Special AI issue.

[Gorti et al., 1993] Gorti, S., Gupta, A., Wong, A., and Sriram, D. (1993). A Model of Integrated Product-Process Representation in Design Synthesis. *IESL Technical Report*. In preparation.

[Grubber et al., 1992] Grubber, T., Tenenbaum, J., and Webber, J. (1992). Toward a knowledge medium for collaborative product development. In Gero, J., editor, *Artificial Intelligence in Design '92*, pages 413–432. Kluwer Academic Publishers, London, England.

[Gupta, 1993] Gupta, A. (1993). Personal Conversation.

[Howard et al., 1989] Howard, H., Levitt, R., Paulson, B., Pohl, J., and Tatum, C. (1989). Integration: Reducing Fragmentation in AEC Industry. *Journal of Computing in Civil Engineering*, 3.

[Kim et al., 1989] Kim, W., Bertino, E., and Garza, J. (1989). Composite Objects Revisited. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Portland, Oregon.

[Kim and Chou, 1988] Kim, W. and Chou, H. (1988). Versions of Schema for Object-Oriented Databases. In *Proceedings of the 14th Very Large Data Bases Conference*, Los Angeles.

[Klein, 1992] Klein, M. (1992). DRCS: an Integrated System for Capture of Designs and their Rationale. In Gero, J., editor, *Artificial Intelligence in Design '92*, pages 393–412. Kluwer Academic Publishers, London, England.

[Klein et al., 1990] Klein, M., Lu, S., and Baskin, A. (1990). Towards a Theory of Conflict Resolution in Cooperative Design. In *Proceedings of the Twenty-Third Annual International Conference on System Sciences*, pages 41–50, . IEEE .

[Kumar, 1992] Kumar, L. (1992). Personal Conversation.

[Kunz and Rittel, 1970] Kunz, W. and Rittel, H. (1970). Issues as Elements of Information Systems. Institute of Urban and Regional Development Working Paper 131, University of California, Berkeley, Berkeley, CA.

[Lander and Lesser, 1989] Lander, S. and Lesser, V. (1989). A Framework for the Integration of Cooperative Knowledge-Based Systems. In Sanderson, A., Desrochers, A., and Valavanis, K., editors, *Proceedings of IEEE International Symposium on Intelligent Control*, pages 472–477, Albany, NY. IEEE.

[Lee, 1990] Lee, J. (1990). SIBYL: A Qualitative Decision Management System. In P. Winston and S. Shellard, editor, *Artificial Intelligence at MIT: Expanding Frontiers*, chapter 5, pages 104–133. MIT Press, Cambridge, MA.

[Loucks, 1989] Loucks, D. (1989). Analytical Aids to Conflict Management. In Viessman Jr., W. and Smerdon, E., editors, *Managing Water-Related Conflicts: The Engineer's Role*, pages 23–37. ASCE.

[Moses, 1987] Moses, J. (1987). Organizing for Change. In *Xerox-MIT workshop on Visions of Design Practices for the Future.*

[Myers, 1992] Myers, J. (1992). Constructive Resolution of Construction Industry Disputes. In Fred Moavenzadeh , editor, *Proceedings of the Conference on the Construction Industry in the Northeast: Opportunities for the 21st Century*, pages 103–104, Boston, MA. Center for Construction Research and Education .

[Object Design, Inc., 1991] Object Design, Inc. (1991). *OBJECTSTORE User Guide.* Object Design, Inc., Burlington, MA. Release 1.1 for Unix-Based System.

[Ostrofsky, 1977] Ostrofsky, B. (1977). *Design, Planning, and Development Methodology.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

[Pahl and Beitz, 1988] Pahl, G. and Beitz, W. (1988). *Engineering Design - A Systematic Approach.* The Design Council, London, UK.

[Peña Mora et al., 1994] Peña Mora, F., Logcher, R., and McManus, T. (1994). SCHEREC: SCHedule RECovery system. In *Proceedings of the 1st ASCE Congress on Computing in Civil Engineering*. ASCE.

[Potts and Bruns, 1988] Potts, C. and Bruns, G. (1988). Recording the Reasons for Design Decisions. In *Proceedings of the 10th International Conference on Software Engineering*, pages 418–427. IEEE.

[Project, 1994] Project, C. (1994). Secretary Kerasiotes Selects River Crossing Design. *Artery Express*, Winter:1 and 6.

[Quillian, 1968] Quillian, M. (1968). Semantic Memory. In Minsky, M., editor, *Semantic Information Processing*. MIT Press, Cambridge, MA.

[Raphael, 1968] Raphael, B. (1968). SIR: A Computer Program for Semantic Information Retrieval. In Minsky, M., editor, *Semantic Information Processing*. MIT Press, Cambridge, MA.

[Rossignac et al., 1988] Rossignac, J., Borrel, P., and Nackman, L. (1988). Interactive Design with Sequences of Parameterized Transformations. In *Intelligent CAD Systems 2: Implementational Issues*. Springer-Verlag, New York, NY.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

[Serrano, 1987] Serrano, D. (1987). *Constraint Management in Conceptual Design*. PhD thesis, Massachusetts Institute of Technology.

[Sheridan, 1993] Sheridan, J. (1993). Personal Conversation.

[Souder, 1988] Souder, W. (1988). Managing Relations between R&D and Marketing in New Product Development Projects. *ComputeJournal of Product Innovation Management*, 5:6–19.

[Sriram, 1988] Sriram, D. (1988). Intelligent Systems for Engineering: Knowledge-based and Neural Networks. Technical report, IESL, MIT.

[Sriram et al., 1994] Sriram, D., Gupta, A., Wong, A., Vemulapati, M., Gorti, S., Fromont, B., Su, V., and Vaidya, V. (1994). An Object-Oriented Knowledge Based Building Tool for Engineering Applications. IESL Technical Report IESL-91, MIT.

[Sriram and Logcher, 1993] Sriram, D. and Logcher, R. (1993). The MIT Dice Project. *Computer*, 26(1):64–65.

[Sriram et al., 1989] Sriram, D., Logcher, R., Groleau, N., and Cherneff, J. (1989). DICE: An object-Oriented Programming Environment for Cooperative Engineering Design. IESL Technical Report IESL-89-03, MIT.

[Sriram et al., 1991] Sriram, D., Wong, A., , and He, L. (1991). GNOMES: An Object-Oriented Non-manifold Geometric Engine. IESL Technical Report IESL-91, MIT.

[Stefik and Bobrow, 1986] Stefik, M. and Bobrow, D. (1986). Object-Oriented Programming: Themes and Variation. *AI Magazine*, 6(4).

[Sycara, 1989] Sycara, K. (1989). Cooperative Negotiation in Concurrent Engineering Design. In D.Sriram, Logcher, R., and Fukuda, S., editors, *Proceedings of MIT-JSME Workshop in Computer-Aided Cooperative Product Development*, pages 269–297, Cambridge, MA USA .

[Thompson and Lu, 1990] Thompson, J. and Lu, S. (1990). Design Evolution Management: A Methodology for Representing and Using Design Rationale. In *Proceedings of the Second International ASME Conference on Design Theory and Methodology*. ASME, ASME.

[Toulmin, 1958] Toulmin, S. (1958). *The Uses of Argument.* Cambridge University Press, Cambridge, England.

[White et al., 1972] White, R., Gergely, P., and Sexsmith, R. (1972). *Structural Engineering.* John Wiley & Sons, Inc., NY, USA.

[Will, 1991] Will, P. (1991). Design Information Handling. In *Proceedings of the NSF Workshop on Information Capture and Access in Engineering Design Environments,* pages 25–34, Ithaca, NY.

[Winston, 1984] Winston, P. (1984). *Artificial Intelligence.* Addison-Wesley Publishing Co., Inc., Reading, MA, second edition.

[Winston and Horn, 1984] Winston, P. and Horn, B. (1984). *LISP.* Addison-Wesley Publishing Co., Inc., Reading, MA, second edition.

[Wong, 1993] Wong, A. (1993). *SHARED Workspaces for Computer-Aided Collaborative Engineering.* PhD thesis, Massachusetts Institute of Technology.

[Wong and Sriram, 1993a] Wong, A. and Sriram, D. (1993a). An Extended Object Model for Design Representation. *IESL Technical Report.* In preparation.

[Wong and Sriram, 1993b] Wong, A. and Sriram, D. (1993b). SHARED: An Information Model for Cooperative Product Development. *Research in Engineering Design.* Fall 1993.

[Woodson, 1966] Woodson, T. (1966). *Introduction to Engineering Design.* McGraw-Hill, Inc., New York, NY.