AUTOMATING NEGOTIATED DESIGN INTEGRATION: FORMAL REPRESENTATIONS AND ALGORITHMS FOR COLLABORATIVE DESIGN

by

WILLIAM N. ROBINSON

A DISSERTATION

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 1993

Approved: [illegible]

Dr. Stephen F. Fickas

An Abstract of the Dissertation of

William N. Robinson for the degree of Doctor of Philosophy

in the Department of Computer and Information Science to be taken March 1993

Title: AUTOMATING NEGOTIATED DESIGN INTEGRATION: FORMAL REPRESENTATIONS AND ALGORITHMS FOR COLLABORATIVE DESIGN

Approved: [illegible]

Dr. Stephen F. Fickas

This dissertation presents a methodology and automated algorithms for collaborative design. The methodology calls for individuals to independently create designs achieving their own goals, and then collectively derive a single unified design using automated negotiation techniques. From a software engineering perspective, the methodology provides parallelism, simplicity, rationale, and reuse. From a negotiation perspective, the methodology provides multiple agent preference maximization and novel resolution synthesis. From an artificial intelligence perspective, the algorithms provide automation for the complex processes of conflict detection, resolution synthesis, and resolution selection. This dissertation describes how the selfish interests of individuals or subgroups can productively aid the derivation of robust collaborative designs through the automated negotiation of their conflicts.

This dissertation describes formal representations for modeling individual perspectives, design conflicts, and subtasks involved in negotiation. Specifically described are representations for: (1) goals and preferences over domain operators, objects, and relations, (2) categories of design and goal conflicts, and (3) categories of conflict resolutions. Automated processes can manipulate these representations to aid group negotiation.

This dissertation describes formal algorithms for detecting conflicts and synthesizing resolutions. Specifically described are algorithms for: (1) distinguishing between simple design differences and design interference, (2) mapping between goals and their supporting design components, (3) detecting goal conflicts, (4) synthesizing analytic and heuristic resolutions, and (5) reintegrating resolved goals into a design. Analytic resolution consists of compromise generation using a multiple criteria linear programming method. Heuristic resolution consists of search through domain hierarchies to synthesize dissolutions and compensations. These methods have been implemented and applied.

This dissertation describes the implementation of our negotiation algorithms and their application to library design problems. The design of library systems is a complex, multiple agent, negotiation enterprise. We have represented portions of documented library designs in our implemented collaborative design tool, Oz. Oz has been used to detect conflicts and derive negotiated resolutions similar to those published by expert librarians. The implementation and its application to the library domain support the central tenet of this dissertation: processes of negotiated design can be automated through the representation of a generic domain model and specific representations of individual perspectives.

VITA

NAME OF AUTHOR: William N. Robinson

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Oregon State University
Mt. Hood Community College

DEGREES AWARDED:

Master of Science, 1987, University of Oregon
Bachelor of Science, 1984, Oregon State University

AREAS OF SPECIAL INTEREST:

Artificial Intelligence
Software Engineering

PROFESSIONAL EXPERIENCE:

Teaching Assistant, Department of Computer and Information Sciences, University of Oregon, Eugene, 1991-1993.

Research Assistant, Department of Computer and Information Sciences, University of Oregon, Eugene, 1986-1991.

Teaching Assistant, Department of Computer and Information Sciences, University of Oregon, Eugene, 1985-1986.

Research Assistant, Department of Computer and Information Sciences, University of Oregon, Eugene, 1984-1985.

Programmer, Battelle Northwest Laboratories, Richland, Washington, Summer 1984.

NORCAS Fellow, Battelle Northwest Laboratories, Richland, Washington, Summer 1983.

PUBLICATIONS:

Robinson W.N., *Automating the parallel elaboration of specifications: preliminary findings*, CIS-TR-89-02, University of Oregon, 1989.

W.N. Robinson, *A decision theoretic perspective of multiagent requirements negotiation*, In Automating software design: interactive design: Workshop notes from the ninth national conference on artificial intelligence, AAAI, July 15, 1991, 154-161. (Also available as RS-91-287 from ISI/USC.)

M. Feather, S. Fickas, W. Robinson, *Design as elaboration and compromise*, Proceedings of the AAAI-88 Workshop on Automating Software Design, Kestrel Institute, AAAI-88, St. Paul, MN, August 25, 21-22, 1988.

S. Fickas, J. Anderson, W.N. Robinson, *Formalizing and automating requirements engineering*, CIS-TR-90-03, University of Oregon, April 6, 1990.

Robinson W.N., *Integrating multiple specifications using domain goals*, 5th International Workshop on Software Specification and Design, IEEE, 1989, 219-226. (Also available as CIS-TR-89-03 from the University of Oregon.)

W.N. Robinson, *Negotiation behavior during requirement specification*, Proceedings of the 12th International Conference on Software Engineering, IEEE Computer Society Press, Nice, France, March 26-30 1990, 268-276. (Also available as CIS-TR-89-13 from the University of Oregon.)

W.N. Robinson, S. Fickas, *Negotiation freedoms for requirements engineering*, CIS-TR-90-04, University of Oregon, April 6, 1990.

W.N. Robinson, *Negotiation in composite system design*, CIS-TR-91-11, University of Oregon, May 1, 1991. (Presented at Stanford Spring Symposium, March 26-28, 1991).

W.N. Robinson, *Preference and function modeling in requirements mediation*, In Model-based reasoning: Workshop notes from the ninth national conference on artificial intelligence, AAAI, July 14, 1991.

Fickas S., Downing K., Novick D., Robinson B., *The specification, design and implementation of large knowledge-based systems*, IEEE Northwest Conference (NORTHCON), Computer Science, Portland, OR., October, 22-24, 1985, 8/3.

Robinson W.N., *Towards the formalization of specification design*, Masters thesis, University of Oregon, June 1987.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the help of my advisor (Stephen Fickas), committee members (Bob Clemen, Art Farley, Gary Meyer), KATE research group members (John Anderson, Anne Dardenne, Brian Durney, Rob Helm), and outside researchers (Stephen Easterbrook, Martin Feather, Anthony Finkelstein, Lewis Johnson). Specifically, I thank Stephen Fickas for his generous support and guidance; Bob Clemen for his patience in explaining decision theory; Stephen Easterbrook and Anthony Finkelstein for their insightful discussions comparing our research; and Martin Feather for his originating research.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This dissertation describes the automation of negotiation processes within the context of group design. Specifically, it describes: (1) a collaborative design methodology predicated on negotiation, and (2) supportive algorithms which automate negotiation processes. This dissertation argues for the effectiveness of the methodology and its automated support.

This research lies within the field of group design since it describes how to structure the process of articulating and formalizing group goals. It lies within the field of artificial intelligence (AI) since it describes how to automate an aspect of human behavior. Finally, it also lies within the field of decision science since it describes, formalizes, and supports decision procedures.

This dissertation provides a key link in the automated assistance of group design interaction. Its most basic assumption is that group members engage in negotiation. Using this, I demonstrate that one can create a mechanism which automates portions of negotiation. This claim itself has been shown by others using different mechanisms (e.g., analytic procedures). However, this research departs from others in its combination of negotiation mechanisms. Rather than exclusively use numeric compromising or heuristic modification, it combines compromise, specialization, and generalization via an interactive resolution search procedure. I show that this approach can: (1) automate portions of negotiation, and (2) assist groups in deriving adequate resolutions.

While negotiation automation is the heart of this dissertation, its impact upon group design is of equal concern. The negotiation algorithms were not designed in a vacuum. Rather, they were developed to support group design. I argue that the existence of an effective automated negotiation system for design will enable a new paradigm for group design. Heretofore, independent design followed by integration has been intractable, not only because of the bookkeeping involved, but also because of the complexity of the task and the limited number of experts qualified to derive resolutions. I present a new design methodology predicated on the existence of effective negotiation support and use decision science arguments to show why such a methodology can better the designs of a more centralized design methodology.

In this dissertation I consider what qualitative difference may be obtained from a design methodology predicated on negotiation. Since this is the early stages of a new paradigm, experimental evidence of its use cannot presented. Instead, I present my group design methodology, called *Multiple Perspective Design*, and argue for its effectiveness using software engineering and decision science arguments. After establishing the usefulness of a negotiation based methodology, algorithms to support negotiation within a group design context are presented. I argue that given a formal set of goals and preferences, a planning system can be used to produce satisfactory designs, and more importantly, its derivation can aid in the negotiated integration of conflicting designs. Finally, I demonstrate the effectiveness of the negotiation algorithms by applying them to portions of documented library design problems and compare the results against documented solutions. Unfortunately, documented designs often leave negotiations implicit; hence, I have interpreted prior traces with regard to negotiated decisions. Additionally, I present the critiques of an expert to confirm the effectiveness of the resolution mechanism.

The remainder of this chapter introduces conflict (§1.1) and resolution synthesis (§1.2) in the context of design negotiations. Next, I argue that negotiation is an appropriate area of study for designers (§2.1) since its use will provide significant contributes for design (§2.2). Then, such a contribution is illustrated with a brief automated design negotiation (§3). Finally, this chapter closes with a summary (§4) and dissertation outline (§5).

1. Design Negotiations

Design conflict has both positive and negative effects. It can reduce the quality and timeliness of designs. Conversely, it can increase: (1) problem comprehension, (2) solution alternatives, (3) solution creativity, and (4) solution acceptance[71]. One must manage conflict to reduce its negative impact while maintaining it virtues. Automated negotiation supports this goal. To provide an intuitive understanding of how it can, a library design conflict is presented, followed by a description of my resolution synthesis method.

1.1. Design Conflicts

Collaboration among multiple, independent agents is a pervasive human activity. In the context of design, collaboration is used to derive a consensus about issues, goals, and decisions. Unfortunately, collaborative design raises difficulties: agents have multiple and conflicting goals; finding an acceptable (not even optimal) compromise in what can be characterized as a large and complex space of alternatives is extremely problematic.

As an example, suppose that a group of designers attempt to design a new artifact, say a university library. The "designers" might be broken into technical staff, users of the artifact, managers, accountants, and even lawyers. Each person has his or her own perspective on what the final library should look like and how it should function. Some people may be in direct conflict with each other. For instance, patrons may like 24 hour service and extensive on-line search. In contrast, library administrators worry about holding costs down. Other groups may seem to be in accordance — faculty and students both want access to library resources — but a more detailed analysis may show them also to be in conflict; each has different uses of resources, and hence different requirements. In essence, libraries, as other real world artifacts, are often encumbered with complex design trade-offs and compromises among interested parties. Tools that support collaborative design should explicitly represent and reason about conflict and compromise. Perhaps just as important, modern design tools should record the history of compromise and trade-off that leads to the production version of an artifact, and make this history available to maintainers of the artifact.

There have been various attempts to provide computer support for collaborative design. The majority attempt to avoid conflict among participants. They do so by pre-processing the goals or requirements of a problem to remove conflicts before design begins. In this way, any conflict turned up later is assumed to be an error either by one of the participants in carrying out his or her assigned (and non-conflicting) tasks, or by the pre-processing step in removing conflicts. Approaches that use this preventive conflict-removal step are silent on a formal basis of the conflict-removal process itself. However, even if one could devise a formal method of detecting and removing conflicts in the goals/requirements that represent the pre-design state of a problem, the preventive strategy can go awry in other ways: (1) what appear to be inherently conflicting goals are not, i.e., the participants are not aware of all the designs possible and that a nonconflicting design can be constructed, (2) users change their goals or preferences during design, thereby reaching a design which may have been pruned under a preventive strategy, or (3) individuals who construct their designs may raise or lower their commitments, thereby becoming less or more agreeable to pre-design compromises among abstract goals.

In contrast, a small set of models eschew the preventive approach, and allow conflicts to occur as they will. For instance, the gIBIS model of design allows conflicting positions to be stated by participants in a design team[15]. Arguments can be made for and against each. Eventually one is chosen. The Persuader system captures conflicts between labor and management negotiation teams[82]. While not linked to design, it illustrates an alternative to the preventive strategy, that of resolution generation.

From a decision science point of view, unresolved conflict can prevent the satisfaction of goals. When poorly resolved, dissatisfaction and inferior achievement arise, perhaps even accompanied by latent conflicts leading to newly manifest conflicts. Despite this antagonistic relation, conflict can enhance group performance. The early (1930's) traditional view treated conflict as dysfunctional behavior to be avoided. Later (1970's), the behavioral view saw conflict as natural consequence of member interactions. Now, the decision science community sees conflict as positive; researchers encourage the identification and resolution of conflicts, bolstering the generative approach (cf., [71]). The benefits of this approach are improved productivity, satisfaction and solution quality[50].

This research is based on the generative approach — the belief that the key to providing effective support for collaborative design lies not in ignoring or squelching conflict, but in recognizing it as a natural and even useful part of group activities. It promotes a style of group collaboration that is based on the views of individual members. My approach is to allow each member to express his or her goals and preferences uninhibited (uncompromised) by what others might want. The outcome is a set of individual designs or decisions that reflect the selfish views of each member. The challenge, of course, is to integrate selfish designs into a consistent whole.

1.2. Resolution Synthesis

As part of the integration process, one must identify conflicts, characterize conflicts, and synthesize resolutions. The resolution synthesis method is the major contribution of this dissertation. It does not simply present pre-enumerated alternatives. Instead, it synthesizes an alternative from pre-enumerated components. The method is simple, general, and widely applicable.

In nutshell, decisions are represented as choices from sets of alternatives, or domains. Conflicts result from differing choices. The alternatives also form possible resolutions. The representations allows one to view a single choice from a variety of related domains, thereby enriching one's view of resolutions. A more concrete discussion will clarify.

1.2.1. Conflict

Any variation of choice is a conflict. For example, the following is an attribute value conflict:

```
Designer-1:
       on_loan(faculty,resource,loan_period.duration=365)
Designer-2:
       on_loan(faculty,resource,loan_period.duration=14)
```

Designer-1 desires to achieve the relation ON_LOAN. It describes the state where faculty have a resource on loan for a specified loan period duration. Designer-1 specifies that faculty should have a loan period of 365 days, while designer-2 specifies they should have a loan period of 14 days. (DURATION is an attribute of the LOAN_PERIOD object.)

1.2.2. Conflict Characterization

After a conflict is identified, it is characterized. For example, the loan period duration conflict can be characterized as different choices from the domain `number_of_days`. Alternative resolutions can also be taken from the `number_of_days` domain. However, one can also characterize the conflict as a choice from the domain pair (`agent,number_of_days`), where one choice was (`faculty,365`). Doing so allows us to consider a wider variety of resolutions, e.g., (`student,14`) and (`faculty,365`). By recharacterizing a simple conflict over a domain like `number_of_days` to other domains and conjoined domains, one can synthesize a wide variety of resolutions. Using this as a basis, the algorithms of chapter V derive negotiation methods of compromise, dissolution, and compensation.

1.2.3. Synthesis

The resolution synthesis process consists of searching through alternative conflict characterizations. By carefully relating the characterized domains *a priori*, and with the aid of design preferences, one can make resolution search simple and efficient. (In the case of design, the algorithms use functional abstraction to relate domains.) Additionally, the method is general as it only depends on one's ability to describe and relate domains. This framework can be applied everywhere one makes choices and can characterize items within domains (e.g., individual design, group design, etc.); however, it has only been applied to resolution generation during design integration.

2. Research Context

As can be seen by the juxtaposition of the previous subsections, the social problem of negotiation and the technical aspect of resolution generation appear widely separated. However, the two do come together in the negotiated design context. Here, I argue that negotiation research should be part of design and give example of benefits that follow naturally from negotiated design.

2.1. Negotiation is Appropriate

Perhaps the most common overarching criticism of this research concerns its appropriateness. In software engineering circles, it is often implied, if not asked, "Is negotiation automation a relevant and appropriate research topic?" The answer is affirmative, not only for the benefit of software engineering, but also for artificial intelligence and decision science.

To understand the obstacles of software development, one must consider its technical and social context[48, 77]. For example, consider a field study of 17 large projects conducted by Curtis et. al. [17]. They found three major problems which affected the quality or productivity of software development:

(1) the thin spread of application domain knowledge,

(2) fluctuating and conflicting requirements, and

(3) communication and coordination breakdowns.

These problems are endemic to group software design. However, current tools do not address these problems. Perhaps this explains why they have relatively low impact on productivity or quality[6, 13, 94]. Moreover, actual programming is a relatively nonproblematic part of the software life-cycle[58, 84, 85]. "Writing code isn't the problem, understanding the problem is the problem."—p. 1271[17]. Clearly, major improvements in software engineering will come from assisting group interactions during problem understanding, goal formation, and conflict resolution. Negotiation methods are shown to help in just these cases.

Software engineering is difficult because of the varied and interacting demands placed on software. Software systems involve physical, social, and software sub-systems; these sub-systems and their interactions are not well understood[29, 30]. Physical and social sub-systems are quite varied (e.g., weather and people) and vary their needs dynamically. To maximize system effectiveness, software engineering must model and reason about system interactions, design systems meeting varied and conflicting needs, and facilitate the adaptation of systems.

The varied and interacting demands on software often arise from the varied goals it must fulfill. Such goals, or *system requirements*, are often the result of negotiations between user groups. In fact, many of the difficulties of requirements engineering can be traced to the goals of: (1) multiple stakeholders and (2) multiple designers. *Stakeholders* are the source of system requirements; for example, patrons, administrators, and librarians of a library. *Designers* are the agents which produce the system specification. Designers must communicate, coordinate, and integrate their work. Furthermore, because they represent the varied needs of stakeholders, conflicts must be resolved. Clearly, software engineering will benefit from an understanding, if not an automation, of negotiation.

Decision science can assist the negotiations of software engineers and their clients. Decision science does not advocate making decisions for individuals, rather it advocates assisting individuals in making decisions according to their own values. It provides methods for enumerating: individual goals and preferences, means of achieving goals, alternatives that result from applying means, and evaluations for alternatives according to individual preferences. Hence, decision science aids an individual in choosing actions which result in states which are most desired according to his preferences. This research also aids groups which interact to satisfy their goals. Such group decision making can include negotiations over the means by which their goals will be satisfied, as well as what the group goals should be.

This research uses decision science modeling techniques to model stakeholder goals and preferences. It uses decision science negotiation procedures to integrate the conflicting designs. One of these procedures is analytic (i.e., compromise generation) and can be shown to generate nondominated resolutions (i.e., no worse than the best). Other negotiation procedures are derived from characteristics of experts. Heuristics such as: "give things of value to stakeholders who don't get what they expect during negotiation" (i.e., compensation) and "divide conflicting items into subclasses and renegotiate" (i.e., dissolution) are encoded in algorithms using AI techniques. The methods rely on decision science for prescriptive and descriptive accounts of negotiation methods and use AI techniques as a means to operationalize them.

Software engineering must address how the varied desires of stakeholders can be understood and met via managing software component interactions within a software development process. Such knowledge, particularly concerning design integration, can be codified, represented, and effectively applied. This research shows that decision science procedures can be an effective means of applying such knowledge to group design.

2.2. Contributions

Software engineering has promised "power tools" which will reduce the software crisis; these tools automatically do mundane tasks, thereby, leaving the programmer free to consider complex tasks. Additionally, power languages have been promised to ease software construction; these languages have constructs which ease the composition of software from simpler, independently developed components. While partially successful, the software crisis is still with us. I believe that some tasks, heretofore consider too complex to automate, or tasks that simply shouldn't be automated, can and should be. Specifically, group design goals should be up for negotiation throughout the development process. Additionally, automated tools can simplify the negotiation task making it a common part of software construction. In fact, power will come from tools which assist common, yet complex, decisions like requirements negotiations that go on throughout the software lifecycle.

My research contributions are derived from an attempt to make power tools powerful. Heeding the creed, *knowledge is power*, I believe knowledge of negotiation is powerful. The single most important contribution of this research has been expanding the discourse on group design negotiation. More specifically, however, the collaborative design methodology and integration automation algorithms have made in-roads into software automation and decision support. These contributions are listed below:

• Methodology: Design in the Large

The methodology is based on design in the large, as opposed to design in the small. It supports independent distributed design. Design members can actually design components maximizing a subset of goals independent of others. Inter-designer communication can be kept to a minimum, thereby allowing fully parallel development.

- Methodology: Design via Composition

  Design is viewed as composition rather than synthesis. Each designer constructs his design by selecting from cataloged components. The catalog is arranged in an abstraction hierarchy, thereby allowing for abstract design. Designers design down to the level of specificity required by their needs and no lower. In this way, the designer is free from making arbitrary design decisions which typically undermine conflict detection and resolution.

- Automation: Domain Model

  Design decision making requires the identification of goals, identification of means (operators) to achieve those goals, and application of means to identify various design alternatives. The abstraction hierarchies provide a generic model for decision systems. Goals, means, and their relationships are automatically abstracted and stored. Similarly, design alternatives satisfying multiple goals can be automatically derived using preferred means.

- Automation: Hybrid Negotiation Search

  Negotiation is viewed as a interactive search process. An arbitrator is presented conflicting goals and is assisted in searching through a complex space of alternatives defined by three generation methods: compromise, compensation, and dissolution. By weighting goals from each perspective, the arbitrator can direct search toward better resolutions.

The basis of the approach is to allow unconstrained independent design and then apply knowledge-based techniques to recover from conflicts. Using abstract planning methods to model component interactions, design level conflicts are traced up to conflicts between abstract means or even stakeholder goals. Once characterized, conflicts are resolved by goal modification or replanning. The result is that every difference between designs is reconsidered during design integration. Conflicts are resolved to maximize modeled stakeholder preferences. Finally, records are maintained for possible reuse and renegotiation.

3. Introduction to Automated MPD

This section introduces the design methodology and its automation. It describes how individuals' design goals and designs are explicitly modeled. It describes how the integration algorithm automates negotiation to resolve conflicts. However, before presenting specific examples, the overall library domain is presented.

3.1. The Library Domain

Throughout this dissertation, library examples will be presented to illustrate negotiation automation. For most readers, the library domain should be familiar; it has provided other researchers with simple examples[45, 89]. However, the domain does provide some interesting complexity.

Libraries come in many forms (e.g., public, private) and serve a variety of needs. Their charters are typically broad and demanding; a university library may be responsible for[57]:

- providing a collection of information resources which meet most of the needs of the university community;
- organizing, maintaining, and controlling collections; and
- providing bibliographic aids in identifying, locating, and using resources.

From such broad charters, more specific policy guidelines are developed (e.g., collection development[55], circulation policies[4, 56], interlibrary loan policies[49]). Finally, from such policies, specific library procedures are designed[65].

Deriving specific library policies and procedures is a process of negotiation. Librarians (desk, circulation, collections), administrators, and patrons all have a stake in library operation. Fees, fines, loan periods, check-out, and renewal policies all result from placating various stake-holders. Some examples are:

• loan periods

From a patron's perspective, loan periods should be as long as possible; this ensures their ability to enjoy borrowed resources. On the other hand, a circulation librarian desires to ensure equal access of resources to all patrons; hence, shorter loan periods provide higher turnover which enables greater access to a large population of patrons. A variety of loan periods result (e.g., 7 days to indefinite); they can vary according to patron type (e.g., child, student, administrator, librarian); they can be extended (e.g., desk renewal, phone renewal); they can be terminated (e.g., recall, revoked privileges).

• fines and fees

Patrons do not want fines. Librarians use fines only to encourage the prompt return of undamaged materials. Administrators may use fines to raise revenues. Like loan periods, fines vary in magnitude according to patron and material type; fines may also be forgiven. Fees are similar to fines except fees are levied to restrict access or raise revenues rather than to punish.

• information access

Patrons want information without restriction. Librarians wish to assist patron information retrieval. However, the administration must protect the privacy of others. A library that allows a patron to view her own borrowing record, but not that of others is a compromise; it protects privacy, but leaves the possibility for illicit access through misrepresentation (e.g., stolen passwords).

Libraries try to satisfy the conflicting concerns of patrons, staff, and administrators; they employ many mechanisms to deal with both errorful and irresponsible behavior; they involve complex responsibility assignments among agents. Libraries are complex systems which require sophisticated analysis to derive adequate specifications.

> Real libraries are not simple. They involve more than just people, books, and a database. They have policies according to who the borrower is, what kind of book it is, what time of year it is, and, of course, exceptions to all of these policies[89].

In contrast, others view library specification as refinement of a generic database/tracking schema[68,69]. This paradigm is based on: (1) capture and storage of library forms and (2) retrieval and modification of such forms for specification. This paradigm appears to be productive for routine situations; for example, primitive programming tasks such as sorting and searching[70]. However, this paradigm must be extended for more exploratory tasks such as group design, especially for informal domains such a library science. Even where significant forms can be captured, the bulk of the requirements work lies in selection and modification. For tasks such as library design, this means understanding policies, their derivation via negotiation, and their effects on alternative designs. This negotiation-oriented paradigm has been combined with the capture-and-modify paradigm for labor negotiations[83]; it illustrates the need for knowledge beyond data schemas for assistance in complex domains.

The library domain illustrates how an artifact's complexity can result from its environmental interface rather than its internal processing. The actual algorithms used in an automated circulation and inventory system may be simple, but the policies they implement may be the result of complex negotiations between system stakeholders. Hence, system complexity may be due to system design, rather than algorithmic derivation; algorithmic enhancements are simple, but system changes must be negotiated with stakeholders.

3.2. Multiple Perspective Design

Originally, this research on collaborative design was purely practice. It was conceived to automate the merging of independent specification design states[73]. Unlike many specification projects, it did not to automate functional decomposition since its organizational philosophy is to prevent conflicts. (In functional decomposition, members construct modules by maintaining carefully defined boundaries[19].) Early efforts included specification by case-based adaptation[27], gradual elaboration[26], and parallel elaboration[72]. The integration of these approaches lead to this dissertation[29, 30].

The current group design methodology, called *Multiple Perspective Design* (MPD), was derived from Feather's parallel elaboration model[23, 24]. MPD supports the acquisition of multiple (conflicting) system perspectives, the subsequent derivation of designs, and the final interactive process of design integration. It exploits group diversity and creativity by addressing conflict recovery.

The methodology captures aspects of negotiation between system stakeholders. Commonly, system requirements are developed through client interviews; sometimes the interviewees are potential operators of system, but often they are presumptuous managers who impose their own requirements without any user consultation. In any case, multiple agent goals are typically unrepresented in requirements engineering models. Figure 1 illustrates how requirements models typically represent a single consistent requirements set, while the variation of multiagent goals is unrepresented.

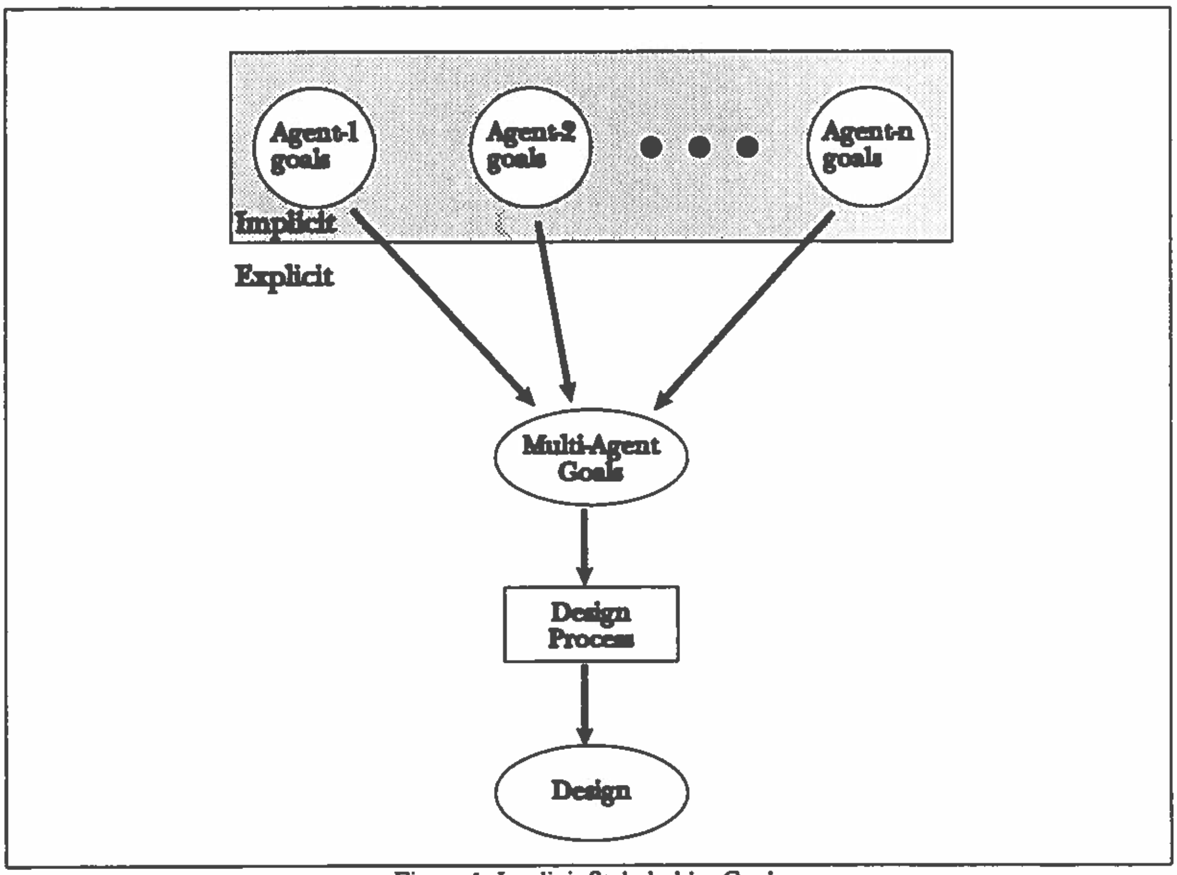


Figure 1. Implicit Stakeholder Goals.

The MPD approach is more direct; it models system participants who might affect or be affected by the proposed system. Mumford calls such an approach participative systems design.

> The argument for a participative approach therefore runs as follows. All change involves some conflicts of interest. To be resolved, these conflicts need to be recognized, brought out into the open, negotiated and a solution arrived at which largely meets the interests of all parties in the situation. Differences of interest will not be confined to management and subordinates but will occur between employees at different hierarchical levels as shown by grading systems, and in different functions. Therefore successful change strategies require institutional mechanisms which enable all these interests to be represented, and the participative design group which consists of representatives of all the different groups in a department will fulfill this function.—p. 112[61].

Mumford applies participative systems design as part of her socio-technical systems design[62]. Her approach is to separately consider social and technical aspects of a system. The social aspect

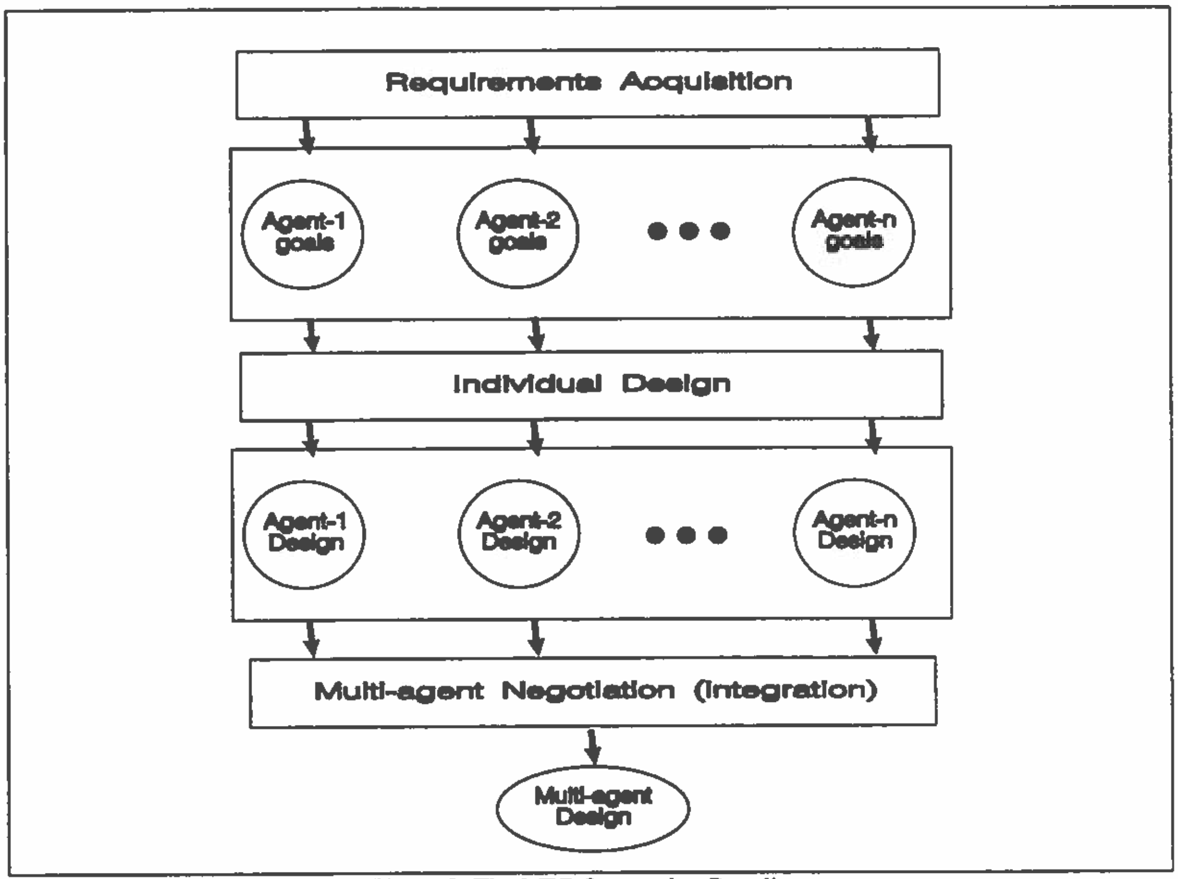


Figure 2. The MPD Integration Paradigm.

principly considered is *job satisfaction*. By way of five attributes, she diagnoses the fit between an employee's expectations and his job requirements. Similar analysis is independently applied between technical structures and their requirements. System alternatives are then evaluated from the social and technical views. Finally, compatible views are combined, followed by selection of the "best" system.

MPD is consistent with soci-technical participative design. Both advocate independent consideration of systems aspects from stakeholder perspectives. However, Mumford's approach restricts independent consideration along two issues: social and technical; whereas, MPD has no issue restrictions. Multiple requirements and multiple designs are also unique to MPD; it is based on the production of a design for each stakeholder. Finally, MPD advocates the integration of "incompatible" designs; the process of their integration reveals the negotiation between stakeholders and can be assisted.

MPD calls for: (1) representing stakeholder beliefs, (2) constructing separate designs for each stakeholder, and (3) integrating designs using negotiation techniques. Figure 2 illustrates the MPD integration paradigm, while figure 3 illustrates where automated support is provided.

System support consists of: (1) agent modeling, (2) development "bookkeeping", and (3) negotiation assistance. A domain model is provided to model available goals, operators, and their interactions. The domain model is requirements language for MPD.

Requirements acquisition consists of using the domain model to construct stakeholder perspectives. Perspectives represent the interests of stakeholders in the proposed system; they are individual's requirements. Acquisition is supported via the domain model and tailoring tools aimed at assisting individual requirements representation.

Design consists of applying the automated planner to individual perspectives. The planner ensures that each requirement is mapped (and linked) to some design components. Tools are also provided for manual editing of designs and managing perspectives and designs.

Design integration consists of conflict detection and resolution generation. For the most part, this process is automated with a human serving to guide resolution search. Presented with conflicting issues and preferences of stakeholders, the human arbitrator actively considers alternative (implicit) preference weightings. Alternatives which appear superior then serve as a focus for incremental improvement by the negotiation operators (compromise, specialization, and generalization). Choosing a resolution ends this interactive process. Finally, an integrated design is output.

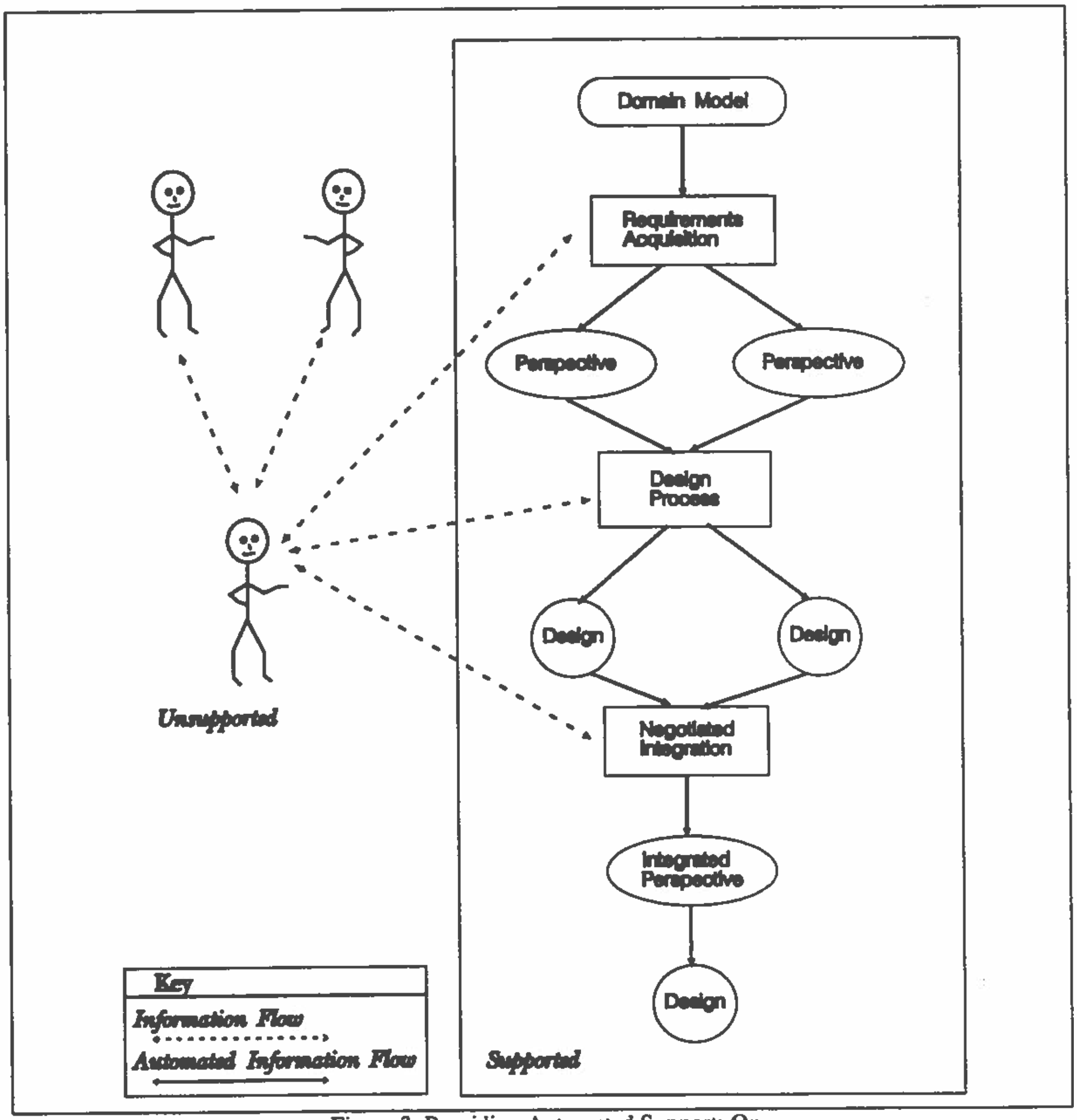


Figure 3. Providing Automated Support: Oz

3.3. A Brief Negotiation

To use the MPD model, Oz is provided to assist individual designers in working independently. Oz is a computer-based system that provides four languages to a designer:

- A language of domain concepts.
- A language for stating goals. These can be goals local to the designer or global to the group.
- A language for stating goal-achievers or operators. The design process itself is one of selecting particular operators for achieving goals that meet the designer's preferences.
- A language for stating preferences. Preferences can be on values (e.g., maximize, minimize), between goals, or between operators.

Each of these languages is based on an abstraction hierarchy. This allows a designer to state abstract goals, abstract operators (and hence abstract designs), and abstract preferences.

3.3.1. Two Perspectives

As an example of goals from the library domain, consider the loan period duration goals of a librarian and patron. Each prefer different values for library loan periods: the librarian perceives a short loan length as beneficial to resource turnover, whereas, the patron perceives a long loan period as a necessity for research. Their preferences of 14 days and 365 days, respectively, can be described as follows:

```
Librarian:
        on_loan(patron,resource,loan_period.duration=14)
Patron:
        on_loan(patron,resource,loan_period.duration=365)
```

Each goal states the desire to achieve a design state where a loan period is granted to a patron. Additionally, the library domains contains patron subtypes, (e.g., faculty, graduate student, undergraduate student). These goals state that for all library patrons, a specific loan period should be granted. Moreover, if the goal cannot be achieved, the nearest feasible substitute should be achieved. For the librarian, this means that designs with loan period durations closer to 14 are preferred over those further away. Similarly, one can state goals in terms of maximizing or minimizing along ordered items (e.g,. `ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD.DURATION=MAX)`). Such preferences can be applied to all modeled domain entities (e.g., goals, objects, and operators).

3.3.2. Two Designs

Given a goal, Oz must find a way to achieve it. Oz represents goal achievement methods as operators. In the library domain, the operators BORROW, RECALL, RENEW and others that allow library users to achieve library usage goals are represented. Given that goals and preferences can be abstract, abstract operators are represented as well. For example, GET_LOAN is an abstraction of the operators BORROW, RECALL, and RENEW. Figure 4 depicts a portion of a library operator hierarchy.

In figure 4, operators are shown as boxes. Each operator has zero or more of the following:

- Consumable preconditions. These are shown directly on the left of an operator. A consumable precondition is one that enables the operator to activate, but is consumed (deleted) in the activation process.
- Persistent conditions. These are shown directly below an operator. A persistent condition is one that must hold to enable a operator. It remains true (it is not consumed) after the operator is activated.
- Produced postconditions. These are shown directly to the right of an operator. A produced postcondition is one that holds (is added) as the result of operator activation.

One of the operators in figure 5, GET_LOAN, is an abstract version of the two operators below it; lines between operators denote abstraction. Abstraction is carried out by finding the set of all

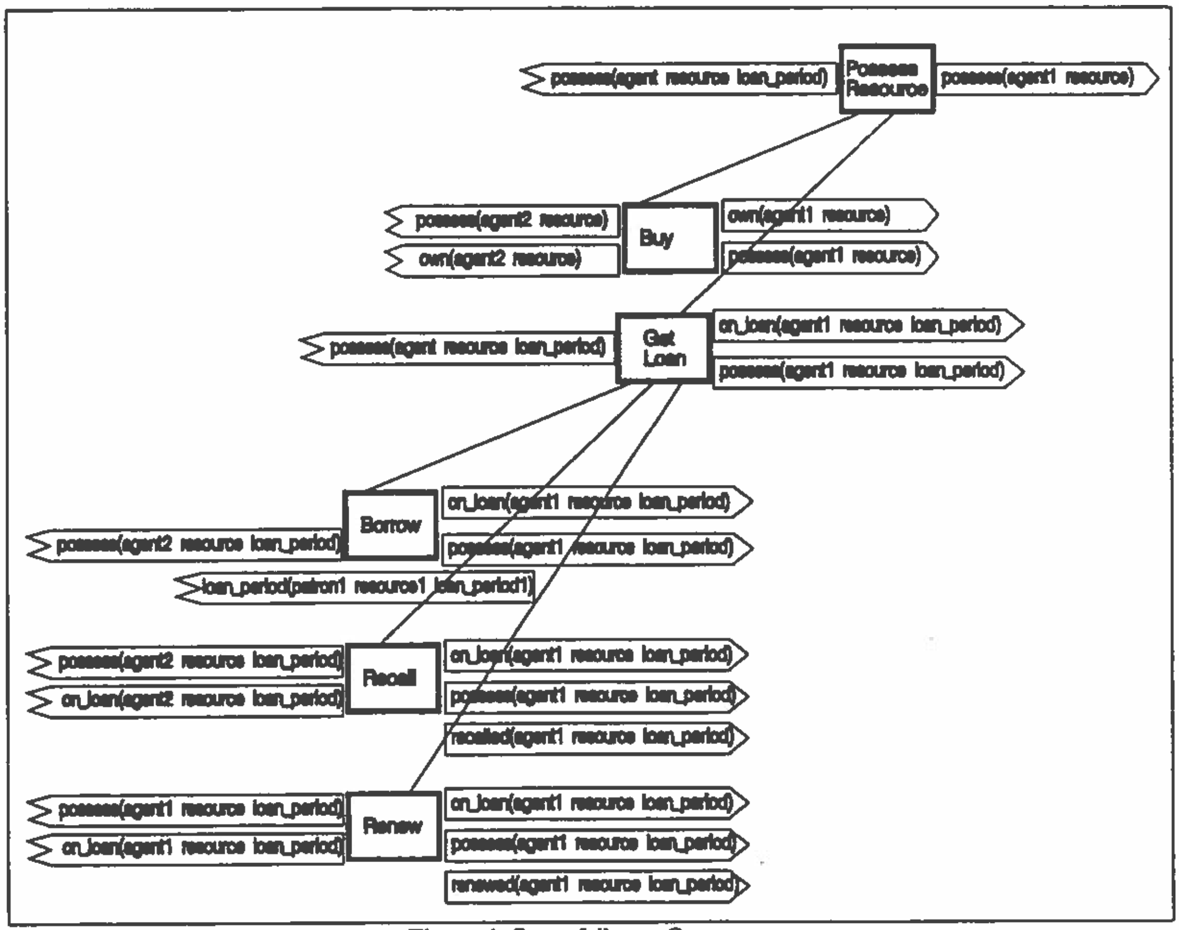

possess(agent resource loan_period)
Possess Resource
possess(agent1 resource)
possess(agent2 resource)
own(agent2 resource)
Buy
own(agent1 resource)
possess(agent1 resource)
possess(agent resource loan_period)
Get Loan
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
possess(agent2 resource loan_period)
Borrow
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
loan_period(patron1 resource1 loan_period1)
possess(agent2 resource loan_period)
on_loan(agent2 resource loan_period)
Recall
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
recalled(agent1 resource loan_period)
possess(agent1 resource loan_period)
on_loan(agent1 resource loan_period)
Renew
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
renewed(agent1 resource loan_period)

Figure 4. Some Library Operators.

unique similarities between two operators, abstract or otherwise, and producing a new, more abstract operator that represents the similarities and eliminates the differences. (This process is carried out automatically in Oz — as new operators are added to the model, they are automatically compared with existing operators. Similarities are detected, and new operators generated and linked into the hierarchy[2].) As part of this process, hierarchies of objects (e.g., patron with subtypes faculty, graduate, and undergraduate) are created. (Additionally, hierarchies of goals can be described manually.)

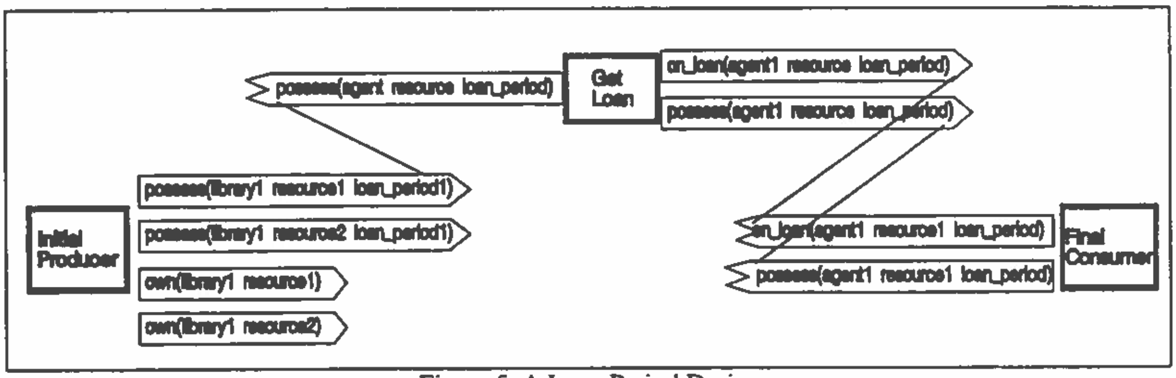
With such library operators, Oz can begin the process of achieving a participant's goals. Goals are achieved by selection of operators. Using a tool called OPIE[28], Oz automates design in the following way:

(1) The most abstract operator O for achieving goal G is found. In the case of the patron's goal, this is the GET_LOAN operator in figure 4.

(2) If there is a preference among the children of O, the most preferred child is chosen next, and step 2 is repeated with the child becoming O. In this example, neither the patron nor librarian expressed any operator preferences.

(3) If no preference is given among the children of O, or if O has no children, then O is selected. Hence, a designer only makes design choices when necessary, otherwise leaving designs in an abstract state. Since the patron wanted the resource for 365 days, the operator GET_LOAN will be instantiated in the design with LOAN_PERIOD's duration attribute set at 365 days.

The above presents the planning process from the patron's perspective. The same process is carried out independently from the librarian's perspective. In this case, the same design is created; however, the GET_LOAN operator is instantiated with LOAN_PERIOD's duration attribute set at 14 days.


possess(agent resource loan_period)
Get Loan
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
possess(library1 resource1 loan_period1)
possess(library1 resource2 loan_period1)
own(library1 resource1)
own(library1 resource2)
Initial Producer
on_loan(agent1 resource1 loan_period)
possess(agent1 resource1 loan_period)
Final Consumer
Figure 5. A Loan Period Design.

3.3.3. A Negotiated Integration

Once the designs are created, one can attempt to integrate them. If there are no difference in the designs, then no negotiation need take place. Otherwise, as in the example, design differences must be characterized as goal conflicts and negotiated.

The integration process consists of:

(1) Conflict Detection and Characterization. Designs differences are identified. MPD applies conflict detection to designs rather than the initial goals because, among other reasons, different goals can lead to identical or non-interfering designs, i.e., designs that differ, but are compatible. In the example, the two GET_LOAN operators assert different values of loan period duration. This interfering difference is characterized as an object attribute value conflict, i.e., a conflict over the value of LOAN_PERIOD's DURATION value.

(2) Conflict Resolution. After the goal conflicts are identified, an individual interacts with Oz's negotiation model to guide the resolution search. First, either value of duration can be chosen, i.e., 14 or 365. Additionally, at the direction of the user, compromise values can be generated (e.g., 189.5). Also, the conflict characterization can be transformed. Conflicts can be generalized or specialized. For example, specializing the conflict characterization suggests the new goal set (among others) of:

```
on_loan(undergraduate,resource,loan_period.duration=14)
on_loan(graduate,resource,loan_period.duration=365)
on_loan(faculty,resource,loan_period.duration=365)
```

In negotiation terms, this resolution can be considered a dissolution, i.e., a removing of the conflict. Moreover, the methods can be combined. Applying compromise to the above specialization can create the following goals:

```
on_loan(undergraduate,resource,loan_period.duration=14)
on_loan(graduate,resource,loan_period.duration=189.5)
on_loan(faculty,resource,loan_period.duration=365)
```

Generalizing conflict characterization is the opposite of specialization. It can be used dissolve conflicts between two similar objects. For example, an object conflict between graduate and undergraduate can be dissolve by generalizing the object of conflict. The following:

```
on_loan(undergraduate,resource,loan_period.duration=14)
on_loan(graduate,resource,loan_period.duration=14)
```

can be recharacterized as:

```
on_loan(patron,resource,loan_period.duration=14)
```

to remove the conflict. The same approach can be used to suggest resource renewal to compensate for short loan periods.

(3) Resolution Implementation. After the analyst chooses a resolution, the system conjoins the original goals with the resolved goals and reapplies the design process. From the example, the new goals are:

```
on_loan(undergraduate,resource,loan_period.duration=14)
on_loan(graduate,resource,loan_period.duration=189.5)
on_loan(faculty,resource,loan_period.duration=365)
```

Since the original goals were transformed to the above goals, only these goals are reapplied to the design process. The resulting design contains three borrow operators which apply to the three different patron subtypes to give three different loan period durations.

The above illustrates how negotiation fits into MPD and the types of negotiations that Oz is capable of automating. All the automation shown above has been implemented (e.g., conflict detection, compromise, specialization, generalization, resolution implementation). In fact, unless otherwise noted, all descriptions of automation and example presented have been automated in the implementation, Oz. The above example also illustrates the mode of user interaction. An Oz user guides the generation and selection of resolutions, while Oz does the mundane work of analytic compromise generation and conflict recharacterization. Oz automates all the tasks presented in addition to providing support for the bookkeeping aspects of independent design and integration.

3.4. Automation Assumptions

To direct the automation efforts, Oz relies on the four assumptions listed below. The first one describes simplifications which allow us to focus on resolution generation. The second and fourth are specific to the MPD design strategy for interactive design systems. The third is a common decision goal; however, it is not clearly demonstrated in many automated negotiation reasoning systems.

(1) Common Languages

It is assumed that one language represents all stakeholder perspectives and designs. A stakeholder's perspective contains abstractions of operators, objects, and relations associated with stakeholder goals and preferences. The design language is a specialization of the perspective language. Both are based on a predicate calculus planning formalism[2]. By using one language, Oz do not address the process of converting between different perspective ontologies[22, 79]. Nor do it address the process of converting between different design languages[42, 63].

(2) Independent Perspectives and Designs

It is assumed that designers independently describe and maintain perspectives and derived specifications. These are ***selfish perspectives***, representing uncompromised stakeholder preferences, and ***selfish designs***, representing uncompromised complete instantiations of stakeholder perspectives. The process of combining all stakeholder perspectives into a single unified perspective is applied only after designs are integrated. Resolution consists of weighting stakeholder preferences in the context of their conflicting designs, and then making a new group perspective, and finally deriving a design. Only those portions of the perspectives that cause conflicting specifications will be reconsidered, and possibly reconciled, during integration.

(3) Optimal Resolution

Within a given search space, it is assumed that an optimal resolution is desired. The arbitrator has the option to expand the search space via introducing more issues. Such expansion can make a previously optimal resolution appear dominated by new resolutions. Nevertheless, once the issues are fixed, an optimal resolution is chosen according to a weighting of stakeholder perspectives.

(4) Interactive Perspectives

It is assumed that stakeholders have bounded rationality[5]. They have limited information and processing abilities. This implies that the specific problem context can have significant influence on an stakeholder's perspective. For example, the stakeholder may become aware of other operators which can satisfy its goals. Conversely, the stakeholder may become aware of constraints which limit goal achievement. In either case, the stakeholder may wish to reconsider its perspective, i.e., its goals and preferences[25]. It is assumed that search, during design and conflict resolution, may uncover new information which is feedback into the stakeholder's decision process which reconsiders designs, constraints, and their effect on goals and preferences[40]. Modification of a stakeholder's preferences is aided according to context. In this way, *interactive perspectives* are supported; perspectives which model stakeholders and facilitate their modification by humans.

Using these assumptions, Oz provides languages by which to represent stakeholder perspectives and algorithms by which to integrate independent designs. The resulting perspective representation language consists of preference annotated abstraction hierarchies. The resulting integration algorithms combine analytic preference maximization with heuristic conflict recharacterization.

4. Dissertation Summary

This dissertation:

(1) presents group design conflict resolution as a problem,

(2) suggests MPD as an appropriate methodology for addressing this problem,

(3) provides a representation and method for describing individual perspectives, and

(4) provides algorithms to integrate designs based on those representations.

This dissertation is evaluated by :

(1) its capacity to represent design goals and preferences,

(2) its ability to generate acceptable resolutions.

Specifically,

(1) the method is applied post-hoc to rederive portions a simple library case-study, and

(2) the resulting case is compared, by an expert, against the original derivation.

This evaluation leads to the following conclusions:

(1) preference modeling and process-oriented decision making can be effectively combined (i.e., the algorithms are effective and generate appropriate resolutions),

(2) group software engineering can benefit from this hybrid approach to conflict resolution (i.e., the post-hoc rederivation did supply adequate resolutions).

5. Dissertation Outline

The remainder of this dissertation is divided into three parts. The first two describe the methodology and its automation, while the last presents it use. Throughout parts one and two, I distinguish between the representational theory, and the current state of the implementation; unfortunately, the confines of this dissertation have not allowed for a complete implementation of the theory. (Everything has been implemented except methods dealing with operator preferences.) Chapter II presents the *Multiple Perspective Design* methodology. Chapters III and V are the heart of the dissertation. They describe stakeholder modeling representations and negotiation-based integration algorithms. Chapter IV describes the use of a planner to automatically derive designs from the stakeholder perspectives. Finally, chapters VI and VII present an example of the system in operation and draw conclusions.

CHAPTER II

MULTIPLE PERSPECTIVE DESIGN

While many groups engage in negotiations, to their disadvantage, few are trained in negotiation techniques. I am interested in making these techniques more widely available. The remaining chapters of this dissertation detail the MPD methodology and its automation in Oz. Before diving into those details, I wish to motivate the use of automated negotiation.

This chapter briefly describe the MPD methodology in general terms. It presents: (1) acquisition and modeling of multiple perspectives, (2) design from multiple perspectives, and (3) integration of multiple designs. Similarly, the negotiated design paradigm is evaluated in general terms. Benefits perceived by two communities are presented: (1) software engineering (parallelism, simplicity, design rationale, reuse) and (2) decision science (preference maximization, technological closure).

This chapter shows that the benefits derived from the MPD methodology can be had by other methodologies. MPD does not have a lock on the benefits of negotiated design. Indeed, many methodologies can benefit from incorporating even simple negotiation techniques into their support. Similarly, the more novel negotiation methods describe here can also be applied in more conventional situations. To show the general applicability of Oz's negotiation methods, this chapter introduces a few example resolutions which, if represented in Oz, could also be automated.

(1) Dissolution: Law of the Sea

The nations of the world must agree on how to mine the international deepsea floor[67].

These undeveloped seafloors contain some of the worlds most abundant mineral resources. Unfortunately, not all nations have the ability to mine the deepsea floor. To resolve this inequality, the nations tenatively agree to a parallel mining system. An international Enterprise will mine on behalf of all nations, while individual nations mine independently and share some of their profits with the Enterprise. Unfortunately, a few nations will still be more effective than the Enterprise. How can one assure the nations that the better sites won't be taken by the more advanced nations? One resolution is based on dividing the conflicting element, the site, into subelements and renegotiate: a requester should pick two sites. Then, the Enterprise can choose which site it will eventually develop while the requester can immediately develop the remaining site.

(2) Compensation: Library Policy

Library patrons and administrators must be agree on a resource loan period[10]. To fully benefit from library resources, some patrons need long loan periods. While administrators support patron resource usage, they must balance the needs of all patrons. With long loan periods, some patrons will keep resources beyond their useful period. Such "idle resources" can be reduced with a short loan period. How can the administration balance the needs of all patrons against the needs of a few patrons? One resolution is based on maintaining a short loan period, but providing another compensating mechanism by which patrons may gain access to resources: a patron should be able to renew their resources. Then, the library reduces idle time, but working patrons can still have access to needed resources.

(3) Compromise: Group Scheduling

Two people desire to meet for two hours. After consulting their calendars, they realize only one of them is free for two hours; however, both have 90 minutes of free time, as well as some single hour slots. How can the two meet for two hours? One resolution is based on simple compromise: the two should meet for 90 minutes, rather than two hours.

The last example is the most common, and weakest, use of negotiation. The first two illustrate more powerful, yet heuristic, forms of negotiation. They go by a variety of names (e.g., lateral thinking, value-focused thinking). These types of negotiation focus on fulfilling goals rather relaxing them to remove conflict. They often include goal reformulation as in the first two examples, but may include lower level repairs, as in the last. Yet, while I advocate the combined use of compromise, compensation, and dissolution, it should be recognize that even simple compromise assistance and recording can aid collaborative design.

1. Methodology

MPD distinguishs between two types of design agents: designers and their chief. Designers create designs which satisfy a subset of requirements. These subsets are chosen to model individual stakeholder's perspectives. Hence, designers have two tasks: (1) modeling individual stakeholder's requirements in terms of goals and preferences, and (2) creating designs fulfilling modeled goals while maximizing modeled preferences. The chief designer has three responsibilities: (1) assigning designers the task of modeling stakeholders, (2) ensuring designers complete their designs, and (3) arbitrating over design integration.

The integration process itself consists of conflict detection, characterization, resolution, and implementation. If designers produce identical designs, then integration simply outputs the design. However, if conflicts are detected, they are characterized in terms of stakeholder goals and preferences. Then, the chief guides the search for a satisfactory, mutually acceptable, group goals. These are implemented, thereby creating a design maximally fulfilling all stakeholder perspectives.

Figure 6 illustrates this process. The system depicted works as follows:

(1) The Stakeholders give the Chief a problem description.

(2) The Chief orders Designers to model the requirements of each stakeholder.

(3) Each Designer develops a model of a stakeholder's goals and preferences. Then, the Design Assistant (i.e., planner) is applied to derive a design. Each design represents a finished design. Designers are allowed to work uninterrupted until completion.

(4) Individual designs are collected by the Collector. MPD currently provides no guidelines for when this should happen. In fact, I suspect designers may wish to engage in a series of intermediate integrations before the final integration. That is, the group may work independently, integrate their perspectives and designs, and then continue working independently. Others have found such *cross-talk* a useful mechanism to control the design divergence.

(5) The Conflict Detector looks for conflicts among the designs. It does this pairwise across all designs. If no conflicts are found, a design is passed along to the Merger. If conflicts are found, they are recorded and then passed to the Conflict Resolver.

(6) The Conflict Resolver provides an interface for a human negotiator, the Chief. The interface provides control over the search process. By weighting individual goals and preferences, the Chief can direct the application of the heuristic modifier and analytic preference maximizer.

(7) Once all conflicts are resolved, the results are passed to the Merger. The Merger outputs the single, unified design.

(8) Finally, in a complete system, a Notifier would provide feedback to the stakeholder's on how their goals and preferences were incorporated into the design. (Even better, such feedback would be provided to the arbitrator during the resolution process.) Also, an Implementor would take the design as input and produce a (software) system. Neither of these processes are automated in Oz.

The key components of this process are the inputs to the resolution process provided by the stakeholder models, and the algorithms that use these inputs to generate resolutions. These are the topics of chapters III and V. Now, an overview of stakeholder modeling, design, and design integration is presented.

The example presented is keep simple so that one may focus on the philosophy of the methodology. From multiple perspectives, concrete designs, and their integration, this section shows how good design characteristics can be derived.

1.1. Modeling Individual Requirements

• Stakeholder preferences are necessary for effective automated negotiation.
• Preference hierarchies are an effective means of representing stakeholder preferences.

MPD simply requires that individuals be able to describe goal states which they desire to achieve. Additionally, it is preferable that they be able to describe alternative goal states, as well as which states are deemed better than others. An *ordering* of items is called a *preference*. Additionally, individuals may be able to state, on a relative *scale*, how much they prefer one item over another. For example, a few goals can be ordered all around the same value, while other goals which bring much less satisfaction are ordered around much lower values. Such description

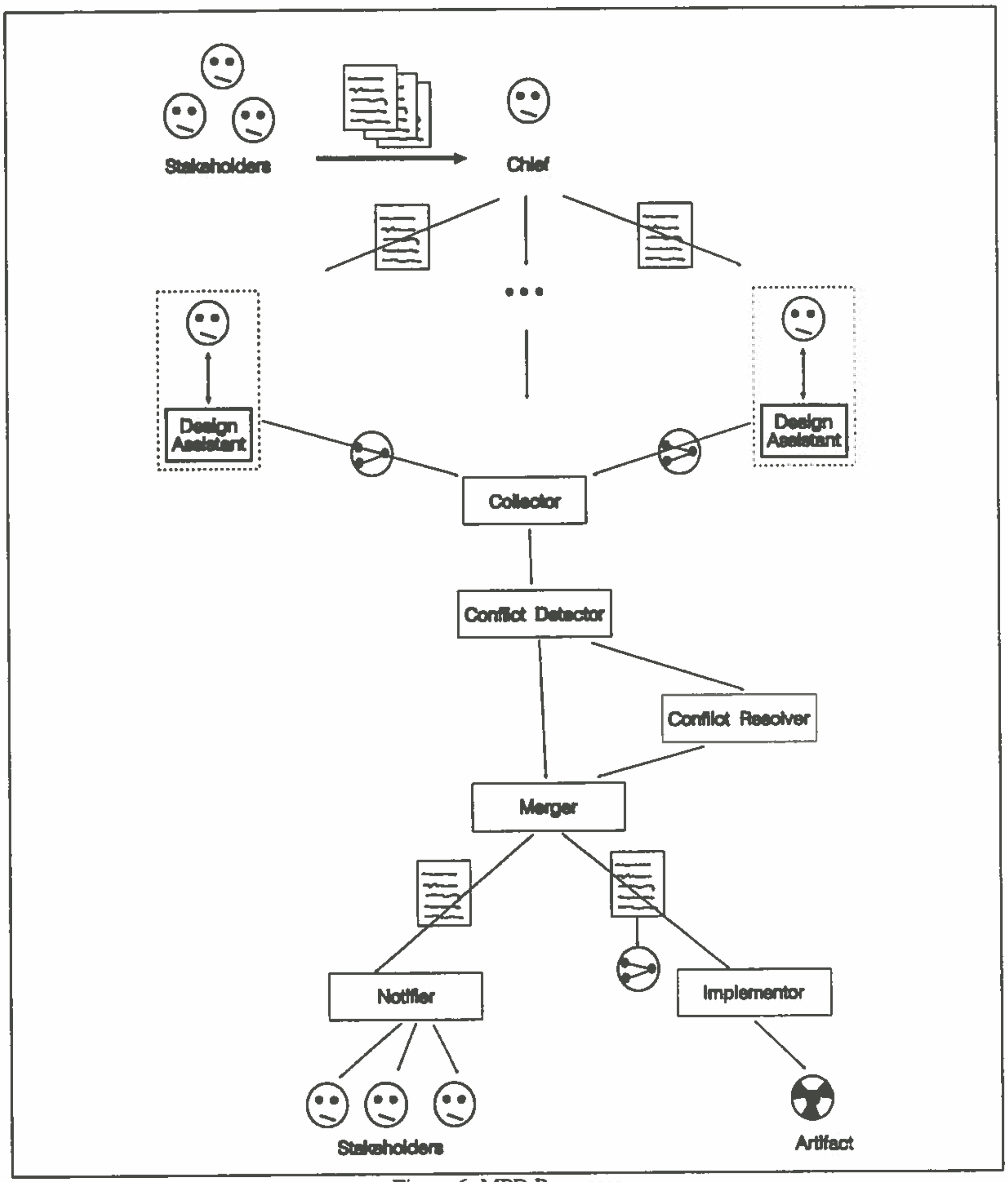


Figure 6. MPD Processes.

provides the negotiation system the means to substitute one item for another. For example, the substitution of one goal achievement for another in the face of conflict. Design objects and operators may similarly be negotiated.

Reconsider the library loan situation of chapter I. The patron may wish to specify his preference over several goals. For example, he may have two goals: OWN and ON_LOAN, and prefer ON_LOAN over owning. To represent this in Oz, the current perspective must first be set (to distinguish the patron's preferences from others). The function, `in-perspective` sets the perspective. Next, the preferences can be described. This is done by modifying a component from one of the domain hierarchies. In this case, the relation hierarchy. The following code fragment illustrates this. It uses the `:order` keyword to describe ACCESS relation preferences. (Relations further down the list, toward the right, are more preferred.) The relations OWN and ON_LOAN are children of the ACCESS relation in the relation hierarchy.

```
(in-perspective 'Patron)

(def_mod_relation access(patron,resource)
  :order '(own(patron,resource)
           on_loan(patron,resource,loan_period)))
```

First note that OWN(PATRON,RESOURCE) describes a state where a patron has bought a resource; similarly, ON_LOAN is a state description, or *relation*. These two relations are specified as specializations of the ACCESS relation. Within the ordered list (defined by the keyword `:order`) they denote the patron's desire to borrow resources, but if that is not available, buy resources. This preference describes the preferred *order* in which goals should be dropped in the face of conflict. Similarly, the patron can order, or *scale*, other entities. For example, the patron could specify that his preference for loan periods falls off linearly from a target value of 180 days:

```
(in-perspective 'Patron)

(def_mod_object loan_period
  :attributes '((duration :min 0 :max 365 :goal 180)))
```

This denotes that the patron desires long loan periods (i.e., around six months). Additionally, this preference is scaled linearly: 180 days is scaled at 100 percent. Periods above and below 180 days will be scaled at lower values. The range of which is defined by the min/max arguments; they are not negotiable, i.e., they must be the same in every perspective.

In addition to goal preferences, individuals may state operator preference. That is, the preferred means by which goals are to be achieved. For example, the patron could specify his preference over several operators:

```
(in-perspective 'Patron)

(def_mod_op get_resource
  :order '(recall, renew, borrow))
```

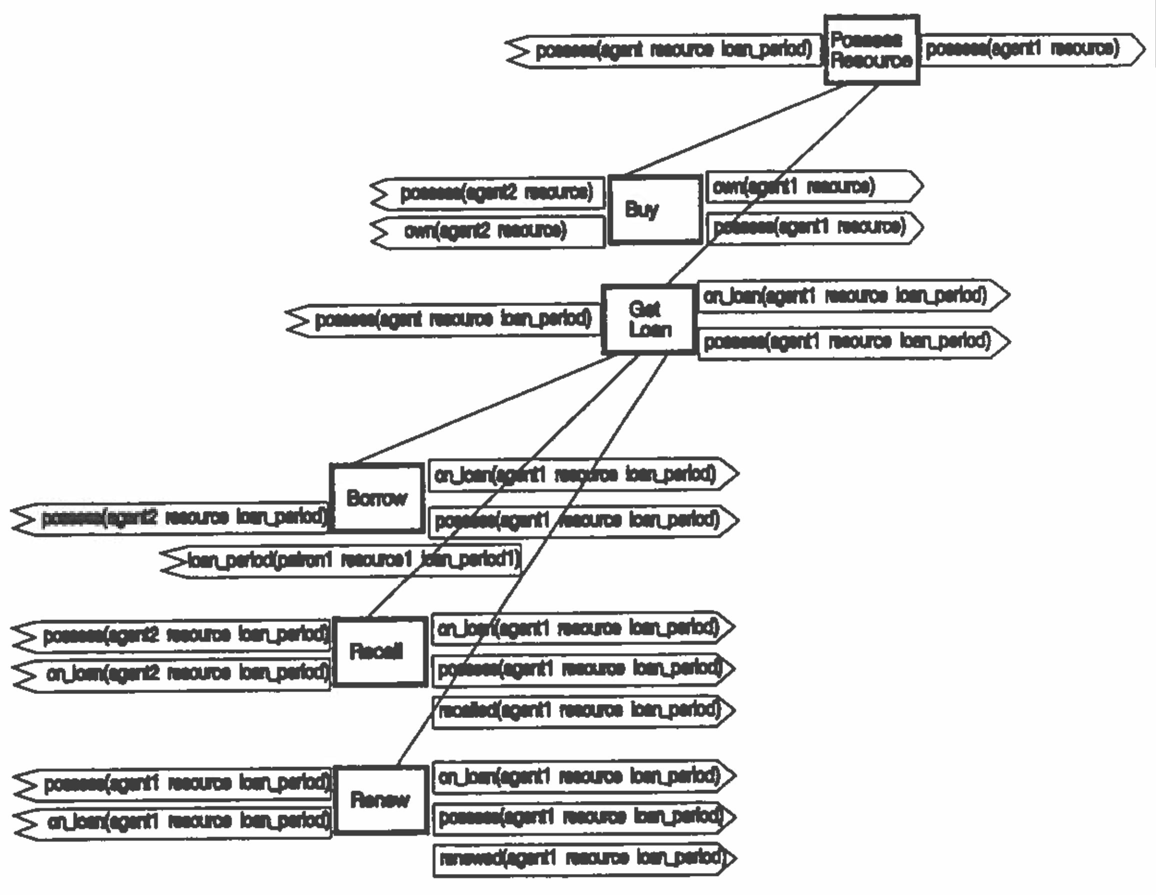
RECALL, RENEW, and BORROW are operators, each of which can produce the ON_LOAN relation. In this way, an individual can state BORROW was the most preferred, while RECALL was the least.[1]

In fact, Oz supports the description of abstract goals, operators, and object preferences. For goals Oz provide for two types of abstraction: (1) relation abstraction, and (2) goal aggregation. Relation abstraction allows one to arrange a hierarchy of relations where lower relations are deemed more specific than those above (e.g., ACCESS is more general than ON_LOAN). Goal aggregation allows the conjunction of multiple relations to be considered as a (composite) goal.

[1]Such preferences can be stated, but their use in negotiation algorithms has not been implemented.

Figure 7 illustrates the hierarchy of operators which produce ON_LOAN. Operators, objects, and relations are stored in such taxonomic abstraction hierarchies. In the operator hierarchy, specialized operators form the leaves, while abstract operators are formed representing common added (right side), deleted (left side), or persistent (middle) relations. Such abstraction is the result of pairwise comparison and is automated[2].

Oz employs a knowledge-based approach. Prior to design, it has a cache of operators, objects, and goals. As part of stakeholder preference acquisition, a designer makes a copy of these


possess(agent resource loan_period)
Possess Resource
possess(agent1 resource)
possess(agent2 resource)
own(agent2 resource)
Buy
own(agent1 resource)
possess(agent1 resource)
possess(agent resource loan_period)
Get Loan
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
possess(agent2 resource loan_period)
Borrow
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
loan_period(patron1 resource1 loan_period1)
possess(agent2 resource loan_period)
on_loan(agent2 resource loan_period)
Recall
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
recalled(agent1 resource loan_period)
possess(agent1 resource loan_period)
on_loan(agent1 resource loan_period)
Renew
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
renewed(agent1 resource loan_period)

Operators RECALL, RENEW, *and* BORROW *are specializations of the* GET_LOAN.

Figure 7. Operators which Produce ON_LOAN.

hierarchies and annotates them with stakeholder preference.

Such hierarchies are effective because stakeholders need only state their preferences down to the level of specificity at which they are concerned. Without such abstraction, stakeholders must use concrete descriptions where they have no vested interest. That is, languages without abstraction may require one to state preferences at levels more specific than one is interested in stating. Such languages can produce arbitrary decisions which result in conflict and unnecessary negotiation.

Additionally, hierarchies are effective because they suggest substitute goal achievement in the face of conflict. When stakeholders prefer an abstract goal achievement, any relation below is satisfactory. For example, any specialization of PATRON in the goal ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD), would be acceptable. However, if none of the specializations can be achieved, achievement of a related relation is a reasonable heuristic substitution. For example, the OWN(PATRON,RESOURCE) relation may be appropriate; it can be had by looking through the operator hierarchy.

Through successive specialization during design, members commit to more detailed design preferences. Conversely, abstract commitments may be successively undone and reconsidered during integration. Negotiation methods can use abstraction hierarchies to generate resolution alternatives of varying generality.

In summary, Oz provides a knowledge-base of operators, objects, and goals relevant to a design domain. Stakeholder acquisition consists of associating preferences with these entities. While the *a priori* knowledge-based simplifies design-time work and the abstraction hierarchies make negotiation search more efficient, multiple stakeholder preferences are the essence of negotiated design.

1.2. Specification Design

- Independent design aids individual preference understanding.
- Independent design allows for maximal parallelism.

Given goals as state descriptions and preferences over operators, objects, and goals, it is (relatively) simple for an abstract planner to derive a plan (i.e., a behavior description) to achieve the goals. Goals are mapped to primitive design operators by successively specializing abstract operators, or by inserting and then specializing operators to satisfy goals[2]. Failure indicates insufficient operators or conflicting individual goals. Currently, Oz do not address either of these issues; however, in the case of conflicting goals, it would be simple to apply Oz's resolution mechanism.

Clearly, independent design allows maximal parallelism. Designers need not consult regarding global constraints nor interface boundaries. Each designer has the flexibility to reconsider requirements in the context of the developing design and ensure the stakeholder's preferences are the correct ones and that such preferences, at least in the ideal independent case, are feasible.

Independent design allows stakeholders to see the consequences of their preferences independent of others. Each stakeholder derives a complete design from his own selfish perspective. Thus, a perspective contains a stakeholder's ideal goals and preferences free of group biases[5,25,40]. Any compromises will be derived from conflicts within a stakeholder's preferences, not from conflicts with other stakeholder perspectives. Therefore, negotiated design aids in the accurate acquisition and recording of stakeholder perspectives. This process itself can eliminate conflicts over abstract, infeasible goals.

1.3. Integrating Designs

- Design conflicts are characterized as stakeholder preference conflicts.
- Preference conflicts are resolved by choosing mutually acceptable compromised and/or substituted values.

The following discussion presents the integration of two designs. Again, the emphasis here is not on how integration is done. Instead, just the basic elements are demonstrated. Essentially, integration consists of characterizing design conflicts as conflicting stakeholder preferences. Then, an automated procedure assists deriving mutually acceptable group goals and preferences. Finally, the planner is applied to the integrated perspective to create a single satisfactory design.

The first design is generated from the patron preferences shown previously in section 1.1; they simply denote the desire for six month loan periods. The second design is generated from similar opposing preferences of the librarian: The librarian has the goal of loaning resources and prefers shorter loan periods (since they reduce resource "idle time" and increase resource accessibility). On the other hand, the librarian adds preferences for library risk and cost, the combination of which must be restricted. The librarian's minor variation in scaling loan period is used to indicate his target value of two weeks (i.e., 14 days); his preference falls off linearly in both directions. The added scales for risk and cost describe the desire to reduce loss and damage risks and processing costs associated with borrowed resources. We've made the simple assumption that both rise linearly with the number of days borrowed. Moreover, their combination is restricted using a linear constraint. Unlike goals, constraints are non-negotiable; the final resolution will have limited risk and cost.

The following code fragments represent these perspectives. The first perspective describes the attributes of the LOAN_PERIOD object for the generic library model; these are inherited by the patron and librarian perspectives. Using the keyword :attributes the attributes DURATION, COST, and RISK are described in order and have their ranges defined. Below, using the keyword :att_constraints, linear relations between the attributes are defined. In each case, there are three constants (one for each attribute) followed by a relation (e.g., <=, =, or >=) followed by another constant. The constants are in the same order as the attributes and associate multipliers for the attributes. For example, the first constraint can be rewritten as:

```
1 * DURATION + 0 * COST -1 * RISK = 0
```

This indicates that risk directly increases with duration.

```
(in-perspective 'Library-model)

(def_mod_object loan_period
 :attributes '(("Duration" :min 0 :max 365)
                ("Cost" min 0 :max 100)
                ("Risk" :min 0 :max 100))
 :att_constraints '((1 0 -1 = 0)  ;Direct increase of risk
                    (1 -1 0 = 0)  ;and cost with duration.
                    (0 0.6 + 0.8 <= 365)))
```

The following code fragments describe the patron and librarian preferences. For the patron, they indicate the desire to maximize loan period duration (e.g., :objective max), while indicating indifference concerning cost and risk.

```
(in-perspective 'Patron)

(def_mod_object loan_period
  :attributes '((duration :min 0 :max 365 :objective max)
                (cost :min 0 :max 365 :objective nil)
                (risk :min 0 :max 365 :objective nil)))

(def_mod_relation access(patron,resource)
  :order '(own(patron,resource)
          on_loan(patron,resource,loan_period)))

(def_mod_op get_resource
  :order '(recall, renew, borrow))
```

For the librarian, they indicate the desire to achieve a 14 day loan period duration (e.g., `:goal 14`), while minimizing cost and risk.

```
(in-perspective 'Library)

(def_mod_object loan_period
  :attributes '((duration :min 0 :max 365 :objective 14)
                (cost :min 0 :max 365 :objective min)
                (risk :min 0 :max 365 :objective min)))
```

After the two designs are constructed, integration begins. It is divided into four subtasks: (1) identifying correspondences and conflicts, (2) forming issues, (3) resolving conflicts, and (4) implementing resolutions. While they are not entirely sequential, it's easiest to consider them so.

1.3.1. Correspondences and Conflicts

In the library example, the stakeholders had corresponding entities. Resources, loan periods, and the borrowing operation all corresponded. However, as correspondences are made, two distinct "conflicting" loan periods are discovered. Illustrated below is an abbreviated attribute conflict record for the loan period duration conflict. It describes the conflict as a difference between object attribute values (LOAN_PERIOD.DURATION) used by corresponding operators (BORROW).

```
TYPE:         OBJECT-ATTRIBUTE
DIFF_OP:      BORROW
INTERFERENCE: (RELATIONS:
                  (LIBRARIAN.LOAN_PERIOD.DURATION: 14,
                   PATRON.LOAN_PERIOD.DURATION: 180))
```

The record denotes that two designs differed in a relation asserted by their corresponding BORROW operators; both stakeholders used BORROW to fulfill their loan period preferences. The patron's design contained a six month loan period while the librarian's design contained a two week loan period. Hence, the object LOAN_PERIOD.DURATION is ascribed *issue* status. (An *issue* is a descriptor denoting alternative values in which conflicting values are found.)

1.3.2. Issues

Given the conflict over their loan period values, one could focus entirely on their specific values: six months vs. two weeks. Such a narrow characterization is ineffective; it limits resolutions to the two alternatives. Instead, it is better to satisfy the goals from which the designs arose. To do so, one can characterize conflicts using abstractions, in this case, the range of loan period duration values. This conflict record characterizes the two loan period durations as different choices from the set of loan periods durations. Hence, LOAN_PERIOD.DURATION is ascribed issue status; thus, it requires resolution. Such characterizations focus on goals, or subgoals, rather than their final means of achievement.

1.3.3. Resolutions

Oz implements three resolution generation methods. These methods are also used by human negotiators[66]. *Compromise generation* maximizes multiple goals under linear constraints. *Specialization* recharacterizes the conflicting relation in terms of more specific relations. *Generalization* recharacterizes the conflicting relation in terms of more abstract relations. Both of these method produce resolutions which can be interpreted as types produced by humans. For example, generalization can produce compensation by giving "losing" perspectives alternative goal satisfaction. Similarly, some conflicts can be "dissolved" by distributing conflicting values over a number of more specialized cases. Each of these methods is demonstrated below.

In the example, loan period duration compromises will be generated first. Next, the arbitrator will pick a duration of 14 days and then choose specialization to recharacterize the conflict. Specialization will suggest introducing the patron subtypes of graduate and faculty, thereby allowing two different loan period durations of 14 and 180 days. Finally, compensation will be given to patrons. By generalizing the conflict, book selling will be introduced into the library.

Compromise Application. Given the preferences and constraints, compromise generates nondominated loan durations between 0 and 365 days.

As shown in figure 8, the arbitrator is visually presented with three nondominated alternative loan period durations: 0 days, 97, and 365 days. The display shows four preferences for each alternative; three are the librarian's (L.COST, L.RISK, L.DURATION) and one is the patron's (P.DURATION).[2] The more a preference is shaded, the higher the satisfaction. For example, RISK and COST are at 100% satisfaction when there are no loans (e.g., 0 days).

The arbitrator considers compromises and attempts to raise the satisfaction of important preferences. In so doing, the arbitrator can observe the interactions between multiple (conflicting) goals in context (e.g., raising L.DURATION also raises P.DURATION, L.COST, and L.RISK). After considering various alternatives and understanding the relationships between goals, the arbitrator

[2]These figures are not generated by Oz. They are provided here as a tutorial simplification of the actual figures produced by Oz. For example, the patron's preferences associated with cost and risk are not shown. Actual screen images are shown in the following chapters.

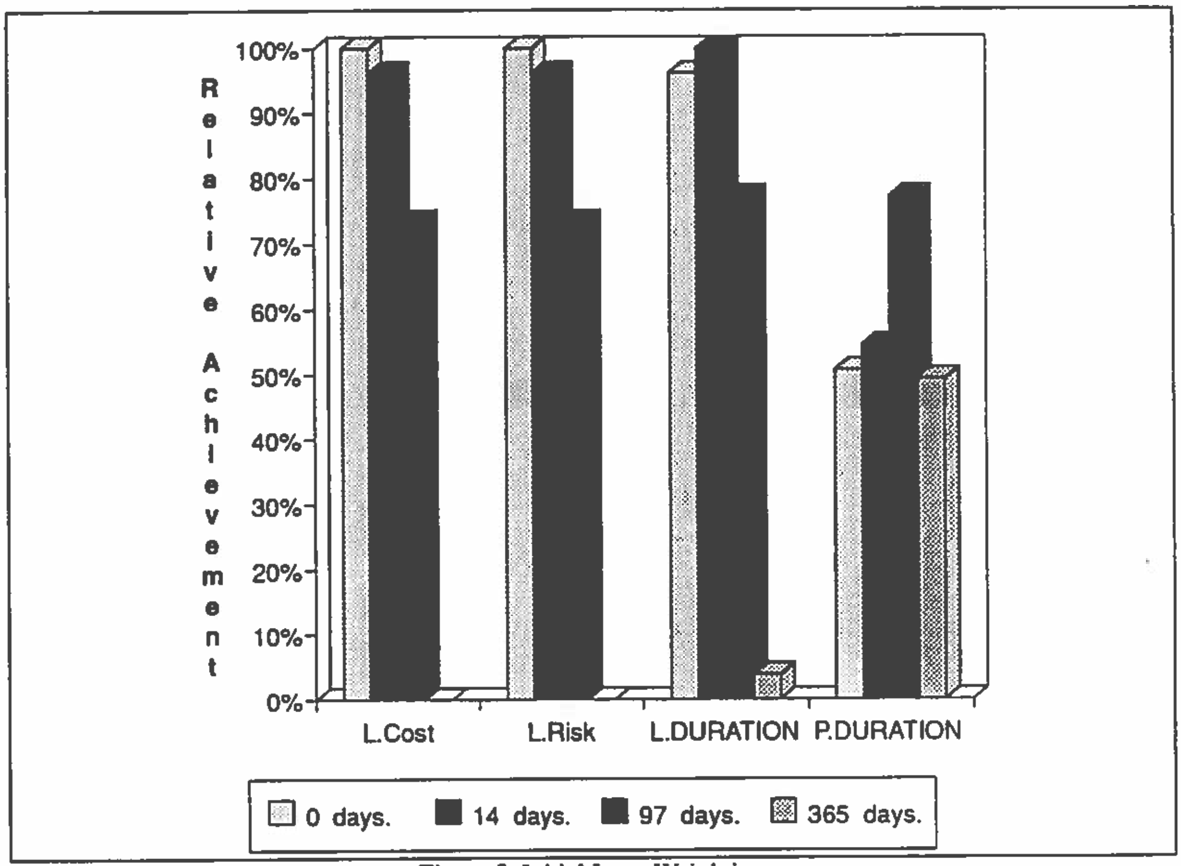


Figure 8. Initial Issue Weighting.

will settle on an alternative. This also indicates an implicit weighting between goals (and between stakeholders); the more satisfaction goals derive from a resolution, the more weighting is implied.[3]

After considering the two extreme solutions and the mean of 97 days, the arbitrator is unsatisfied. He realizes all loan periods are between 0 and 365 days, where risk and cost increase with loan period. With 14 days (the librarian's goal) as the current alternative, he specializes the conflict characterization.

[3]In a more complete implementation, feedback from the stakeholders would be used to derive consensus. Instead, in Oz, this process is outside the automated support.

```
Available (template) Specialization
  1. Relation:             ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD)
     Issue:                LOAN_PERIOD.DURATION
     New Issue/Values:     (GRADS,LOAN_PERIOD.DURATION=14)
                           (FACULTY,LOAN_PERIOD.DURATION=180)
     Use Operator:         BORROW
       Consumes:           GRADS | FACULTY
       Produces:           (GRADS,LOAN_PERIOD.DURATION=14)
                           (FACULTY,LOAN_PERIOD.DURATION=180)
```

Figure 9. An Available Specialization.

Specialization Application. Figure 9 illustrates a single specialization of the loan period conflict. Here it is assumes that two subtypes of PATRON (GRADS and FACULTY) are already represented. Since, these are subtypes of PATRON, BORROW will still apply.

The alternative is displayed in context with a list of issues, values, add objects, and add operators. The alternative suggests that GRADS and FACULTY be consumed by BORROW which should distribute LOAN_PERIOD based on the subtypes. This alternative provides for long loan periods acceptable to patrons, and shorter loan periods acceptable to cost and risk minded librarians. In general, each alternative is displayed with the relation to which it responds. The loan period conflict is the result of interference associated with the ON_LOAN relation; hence, ON_LOAN is displayed to provide context.

The figure notes that alternatives are displayed in template form. This is a reminder that the initial issue/value assignments are arbitrary; e.g., the assignment of GRADS = 14 is the result of mapping previous alternative values or goal values (e.g., 180) to the new objects. Rather than choose among the possible assignments in the specialization procedure, value search is confined to the interactive compromise procedure. Hence, the arbitrator may choose the dissolution (GRADS 14) $\wedge$ (FACULTY 180) and modify this by manipulating a display like that in figure 10.

Figure 10 depicts the new search space. Previous alternatives are displayed, giving the search procedure continuity and context. The preferences for the new issues, GRAD.LOAN_PERIOD

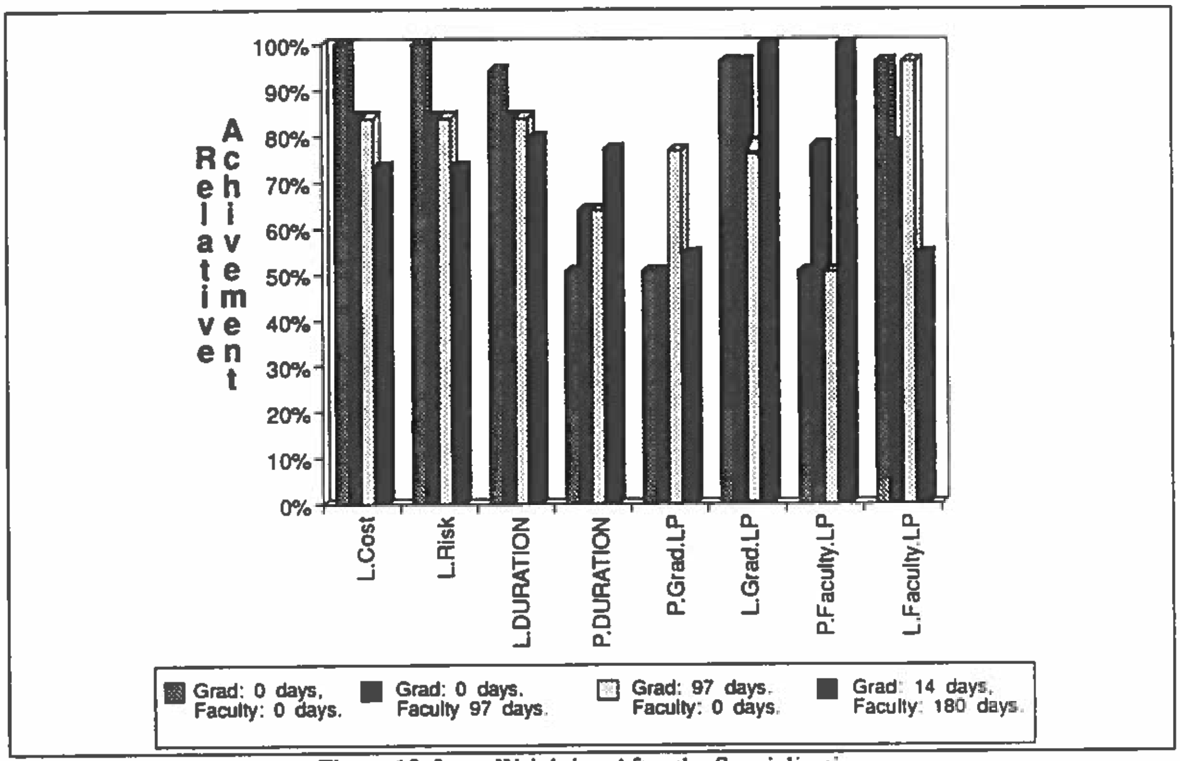


Figure 10. Issue Weighting After the Specialization.

and FACULTY.LOAN_PERIOD, are displayed for the new alternative. Also, since the loan period duration has be decomposed into GRAD.LOAN_PERIOD and FACULTY.LOAN_PERIOD, DURATION now displays the average satisfaction for GRAD and FACULTY.

While the arbitrator considers the new resolution, GRAD 14 $\wedge$ FACULTY 180, a good resolution, he wishes to give the patron compensation for the 14 day loan period. Next, he calls for compensation.

Generalization Application. Figure 11 illustrates hypothetical and available compensations. Observing the operator hierarchy of figure 7, one can see that OWN is a relation added by an operator near the operator that produced ON_LOAN; i.e., BUY is in the same GET_LOAN subtree as BORROW. Since no operator produces OWN in the integration, this alternative is suggested. This first alternative suggests that the BUY operator produce OWN.

The second alternative suggests that a new relation, X, be added as a sibling of ON_LOAN and a new operator be added to produce X. Informally, the first alternative suggests selling resources to compensate for undesirable loan periods, whereas the second suggests the invention of a new type of access mechanism. Currently, only available resolutions are presented in Oz.

The two alternatives can be considered compensation because they achieve a goal similar to one which is unachieveable. The hierarchies are used to determine relevance. Children of the *n*th ancestor may substitute for a relation because they are functionally similar; that is, they are near each other in the operator hierarchy. However, the greater the *n* the less relevant the compensation.

Figure 12 depicts the new search space after alternative 1 of figure 11 is chosen. The preferences L.BUY and P.BUY are displayed for the new alternative. Again, this is a tutorial simplification for preferences which may be attached to the BUY operator (e.g., COST, RISK). (If no preferences were specified, they could be acquired now.) Regardless of BUY preferences, for all loan period alternatives, BUY is unchanged; selling resources does not depend on loan periods and is

```
Available (template) Generalization
  1. Generalize (level 1)
     Relation:              ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD)
     Issue:                 LOAN_PERIOD.DURATION
     New Issue/Values:      OWN(PATRON,RESOURCE)
     Use Operator:          BUY
       Consumes:            PATRON
       Produces:            OWN(PATRON,RESOURCE)
Hypothetical (template) Generalization
  2. Generalize (level 1)
     Relation:              ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD)
     Issue:                 LOAN_PERIOD.DURATION
     New Issue/Values:      X(PATRON,RESOURCE,LOAN_PERIOD)
     Add Relation:          ACCESS->X
     Add Operator:          X_ACCESS
       Consumes:            PATRON,RESOURCE,LOAN_PERIOD
       Produces:            X(PATRON,RESOURCE,LOAN_PERIOD)
```

Figure 11. Hypothetical and Available Generalization.

regarded as good.

Finally, the arbitrator is satisfied with this resolution. The integrated library will sell resources and loan resources for periods of 14 and 180 days. In the final step, an integrated perspective must be produced and then applied to the planner. This is a non-trivial step which involves reasoning about the given designs to properly transform the goals. This is described in chapter V.

1.4. Methodology Summary

Above, a methodology for distributed negotiated design was demonstrated. In doing so, the type and location of automated support provide by **Oz** was shown. The description of the actual algorithms is put off until the following three chapters. Now the benefits of such a negotiation-

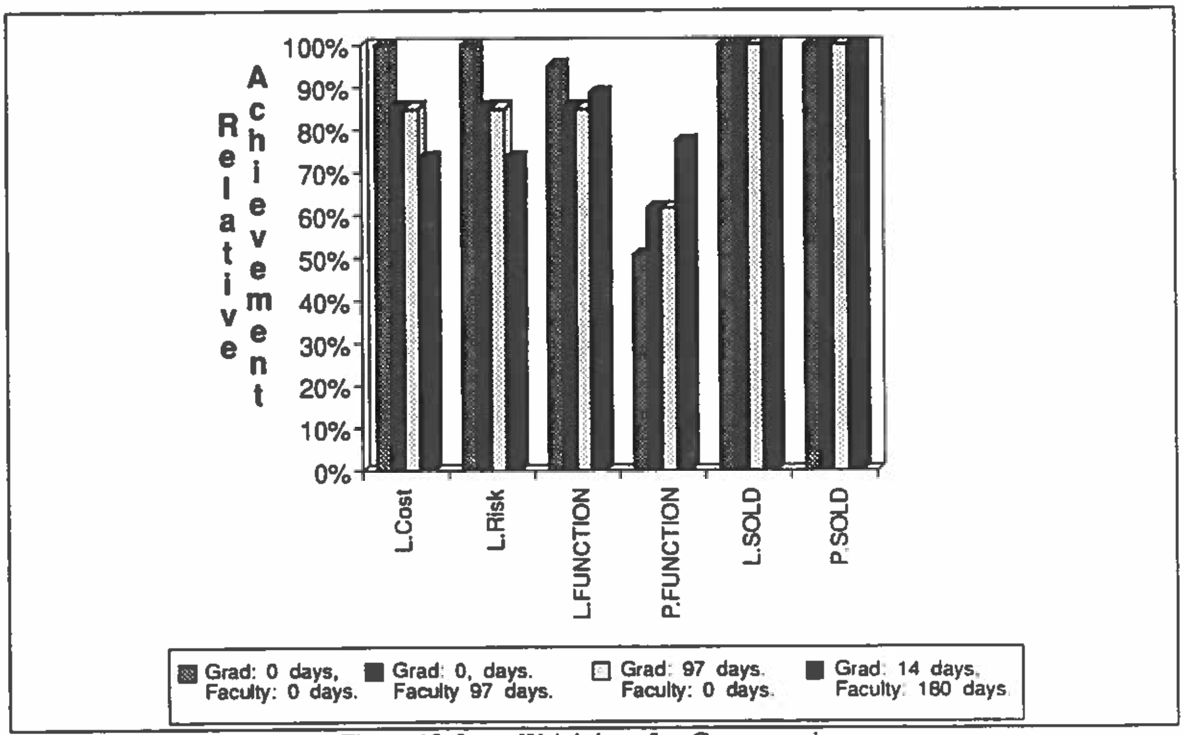


Figure 12. Issue Weighting after Compensation.

based methodology are presented.

2. Evaluation

The MPD methodology is based on the six basic assumptions summarized in figure 13. The following retorical evaluation presents two research perspectives which support these assumptions and the benefits which they derive. Only recently has software engineering research explicitly advocated negotiated design; however, even moderate positions implicitly support negotiated design. Conversely, decision science explicitly provides both methodology and automated means for negotiations; however, their arguments for negotiations also carry over to design. Next, these perspectives are considered in turn.

2.1. A Software Engineering Perspective

Software engineering methodologies are largely concerned with increased productivity and quality. Solutions are typically based on divide and conquer. The design process is decomposed into smaller, independent problems, each of which is solved by a designer, and then composed to

- Stakeholder preferences are necessary for effective automated negotiation.
- Preference hierarchies are an effective means of representing stakeholder preferences.
- Independent design aids individual preference understanding.
- Independent design allows for maximal parallelism.
- Design conflicts are characterized as stakeholder preference conflicts.
- Preference conflicts are resolved by choosing mutually acceptable compromised and/or substituted values.

Figure 13. Methodology Assumptions.

form the final solution. Methodologies vary on methods and orderings of their decomposition and compositions. However, all are intended to (1) increase parallel design activity, (2) simplify individual design. Additionally, some may include mechanisms which provide for: (3) design rationale, and (4) design reuse. Negotiated design provides similar benefits.

2.1.1. Parallelism

Functional decomposition and object-oriented design are examples of divide and conquer methodologies. First, abstract module interfaces are fixed. Next, designers, in parallel, fulfill the interface specifications. Finally, the modules are combined to form the finished software. Such methodologies gain from the rigid interfaces which allow parallel development and compartmentalizes simple errors and their fixes to a subset of modules.

Negotiated design has the benefit of parallel design. The MPD methodology calls for independent design based soley on individual perspectives. Clearly, this allows for maximal parallelism between designers. However, in some cases, designers waste their effort constructing design components which must be removed during integration.

There is a tradeoff between the *a priori* fixed interfaces of modular decomposition methods and the *laissez faire* style of independent design. Fixed interfaces guarantee simple module integration at the expense of design flexibility. Adaptable perspectives allow maximal design flexibility at the expense of module integration complexity. I choose the latter not only for its flexibility, but also for its superior rationale modeling (§2.1.3) and decision support (§2.2).

2.1.2. Simplicity

Simplicity is another argument for compositional methodologies. Large problems are decomposed into smaller problems, each of which is simpler to solve. Experts take advantage of composition. They are observed retrieving abstract templates, instantiating, composing and modifying them to solve problems. Similarly, negotiated design can benefit from composition. Designs can be created via composition of simpler components.

Additionally, negotiated design need only model an individual's perspective. A more traditional method is not so simple. It requires that all individual perspectives be combined into a single requirements model. Subsequently, these global requirements are decomposed into modules for design. In contrast, negotiated design explicitly represents multiple requirements perspectives. Hence, it supports two complementary forms of abstraction: (1) module abstraction, and (2) requirements abstraction. Negotiated design methodologies benefit from the simplicity of compositional design as well as simplified requirements and assisted requirements integration.

2.1.3. Rationale Record

Design rationale provides the ability to answer "how" a design came to exist. Comprehensive design rationale answers "why" a design exists as it does. Rationale may simply contain the sequence of operators applied to arrive at a design. More detailed rationale lists alternatives considered for each design decision along with criteria used to choose among them. The more comprehensive the rationale, the simpler the task of answering "how" and "why" questions.

Often, designs are the result of compromises or other negotiation methods. The methods take multiple, conflicting, perspectives and derive a solution. Often, such solutions make no sense on their own, but must be understood in the context of the negotiations. For example, many cars only have a driver's side air bag. By itself, this makes no sense considering the goal of passenger safety and the associated components of all passenger seatbelts. However, it is easily explained as a compromise between safety and costs; all passenger air bags are deemed too expensive for their perceived benefit. Hence, only a methodology which represents these multiple perspectives and their negotiations will be able to adequately explain such negotiated design components.

2.1.4. Design Reuse

Design reuse provides the ability to use past efforts to solve current problems. In its simplest form, problems are characterized according to parameters. Solutions are derived or retrieved according to their parameters. More sophisticated reuse mixes new goals with past problems. Where applicable, old derivations are used directly or are modified, otherwise new derivations are constructed. Generally, this paradigm is based on: (1) capture and storage of design abstractions and (2) retrieval and modification of such abstractions.

As a design paradigm, reuse has been productive in well understood domains. For example, primitive programming tasks such as sorting and searching[70]. However, this paradigm must be extended for exploratory design. In exploratory design, overlap between design problems is slight. Also, design goals change as solution alternatives become apparent. Hence, even where significant design abstractions can be captured *a priori*, the bulk of the exploratory design lies in modification. For tasks such as library design, this means understanding policies, their derivation via negotiation, and their effects on alternative designs. Therefore, reuse will become more viable as rationale records become more complete, thereby enabling accurate matching and modification of previous abstractions to new situations.

The integration mechanism used in negotiated design provides a possibility for an expanding the role of reuse. Since negotiated design provides the rationale to understand previous designs, old designs may be modified to achieve new goals. Additionally, the integration mechanism may provide a way to combine vastly differing designs (e.g., different domains) to achieve a new goals[74]. This is because in both design and reuse, one must: (1) analyze current design(s), (2) characterize designs in terms of their goal achievement, (3) derive a conflict free goal set, and (4) derive a satisfying design. The integration mechanism of negotiated design provides a single vehicle for both design via integration and reuse via integration.

2.2. A Decision Science Perspective

Decision science is concerned with describing and perfecting decisionmaking methodologies. From a decision science perspective, negotiated design has two important decision characteristics: (1) preference maximization, and (2) technological closure. From these characteristics, one can rederive the maxim: "two heads are better than one". That is, multiple perspectives can produce better decisions than single perspectives. Unfortunately, the reverse can also be true due to social influences. However, if one assumes accurate, honest, cooperative stakeholders, one can apply the following decision theoretic arguments to show that MPD can produce better designs than those from a single perspective.

2.2.1. Preference Maximization

MPD is based on the basic assumption of decision theory: individuals that engage in a rational decision method will produce better results than individuals who do not. A rational decision method for an individual consists of enumerating goals and preferences, identifying alternatives, evaluating alternatives, and choosing the best alternative. Similarly, groups benefit from the enumeration of multiple goals and preferences followed by assisted trade-off analysis.

Groups benefit from the independent description of individual goals prior to negotiation. Simply enumerating individual preferences identifys the decision context and prevents compromised goals before search. That is, adding a perspective can increase the decision context, thereby introducing new alternatives. For example, given the current goal of a two week loan period, one can expand the decision context by introducing a six month loan goal. Even though the two goals may conflict, hence apparently reducing the mutual feasible alternatives to nil, the decision context contains more alternatives. In such a situation, it is the task of negotiation to determine if the two goals do indeed conflict. That is, do the stakeholders really hold mutually exclusive goal, or will they accept both goal states. If the latter is true, then the group benefited from the independent goal descriptions. New alternatives enrich the decision context by providing more choice and allowing decisionmakers to reconsider their preferences. Once all goals and preferences are identified, a decision procedure can identify the best alternative.

Unfortunately, preference enumeration followed by maximization is generally not part of design. Few researchers consider a single set of preferences, hardly any consider multiple conflicting preferences. Such preferences can help explain a design and expand the alternatives. They can also produce novel alternatives.

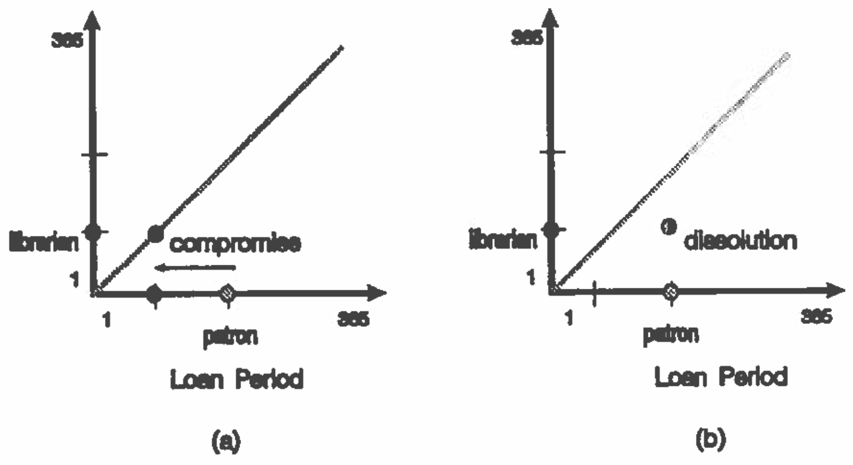
2.2.2. Technological Closure

Innovative designs can be derived using the negotiated design paradigm. First, individuals derive designs from their own goals and preferences. During integration, the group attempts to reconcile conflicting components. Rather than simply choosing one component over another in "tit-for-tat" fashion, individuals demand that their components be included in the final design. To do so, they may be able to conditionalize the way components conflict (e.g., on a contingency basis) to combine the components. In the past, this method has been used to predict new technologies[41]. More recently, it has been suggested as a paradigm for introducing novel alternatives through "dissolution"[92]. This method can assist resolution generation as part of negotiated design.

Design novelty is derived from two sources: individuals and the integration of their designs. First, multiple individuals seeking similar solutions increase the chance that someone will generate a novel design. Second, individually different design fragments can be joined in an integrated design by conditionalizing conflict occurrence[41, 92]. Rather than choose one conflicting design fragment over another, rather than compromise an individual's satisfaction, one can derive a novel design by combining "conflicting" design fragments. Through negotiation, "conflicting" design fragments can be combined to form novel designs.

As an example, consider figure 14. It illustrates perspectives of the patron and librarian. The two axies each represent their preferences over the loan period; preferred loan periods are toward the top right corner. The shaded line through the plane represents both stakeholders choosing the same loan period; hence, it describes obviously *feasible* alternatives. The remaining unshaded region is *infeasible*, since constraints (preconditions) of the loan operator prevent different loan periods from simultaneously being part of a design. The compromise in part (a) illustrates the patron appeasing the librarian to resolve the conflict. Alternatively, the librarian could have appeased the patron by redefining his perspective: by reordering his preferences, the librarian could make six months become the preferred feasible alternative. Best of all, the stakeholders can create a novel design by conditionalizing their constraints. By predicating how "conflicting" entities are to be part of a design, conflicts can be *dissolved*. In the case of two conflicting loan period, one can predicate their use on the type of patron: faculty patrons receive a longer loan period while student patrons receive shorter a loan period. Part (b) of figure 14 illustrates such a dissolution. It represents the integration of the two loan periods.

Multiple preferences aid forming a "technological closure"[41]. Each perspective identifies related criteria associated with an abstract, but apparently infeasible, object. Often, it is a matter of technological ingenuity before the infeasible object becomes feasible. Jantsch used this observation for technological predication. For example, given the introduction of the jet engine during WWII and the existence of airplanes, it became of matter of ingenuity before the flying V2 bomb became feasible. This same method can be used in a generative fashion. Using operator descriptions, one can analyze constraints and predicates to suggest possible fixes to allow infeasible alternatives to become feasible. Hence, multiple conflicting perspectives can induce innovative designs.



In part (a), the patron compromises the preferred six month loan period by accepting the two week period. In part (b), stakeholders dissolve their conflict by conditionalizing constraints, thereby allowing both six month and two week loan periods in the design.

Figure 14. Novelty Induction from Selfish Design.

In negotiated design, individuals seek designs before compromising their perspectives. In contrast, a decomposition methodology compromises perspectives before design construction when perspectives appear to conflict. MPD ignores such *a priori* pruning and seeks designs in any case. This strategy wins over the preventive strategy when: (1) the *a priori* knowledge is wrong (e.g., nonconflicting designs can be constructed), or (2) the *a priori* knowledge becomes wrong (e.g., a novel design is invented which satisfies the "conflicting" perspectives), or (3) stakeholders change their preferences during design, thereby reaching a design which may have been pruned under a preventive strategy[40, 81, 92]. Moreover, individuals who construct their designs may raise their commitments, thereby becoming less agreeable to compromise. However, this may increase the likelihood that they will seek a novel design which satisfies all perspectives.

CHAPTER III

MODELING INDIVIDUAL PERSPECTIVES

Stakeholder goals and preferences must be formally represented before an automated system can reason about them. In the **Oz** representations, a goal is first represented as a choice of an item from a domain, while a preference is an ordering of items within a domain. Hence, to represent goals and preferences, one must represent domains of items. In Oz, items are operators, objects, and relations used to represent the area of concern. A set of such domains used to characterize an area of study is called a *domain model.*

To represent individual perspectives, one: (1) creates a generic perspective, or domain model, (2) creates copies of the domain model for each individual, and (3) specializes each copy by attaching each individual's goals and preferences. After individual designs are constructed to satisfy the perspectives, the designs are integrated. Through the design integration process, the group requirements are created. Finally, a group design is constructed.

Figure 15 illustrates this *Multiple Perspective Design* process. The figure shows a screen image of the Oz design tool. Rectangles containing the five geometric shapes depict domain models. The internal shapes represent, from left to right, initial conditions, operators, objects, relations, and goals. Circles linked below a domain model depict a design created from the model. Designs which link into a trapezoid depict an integration; the trapezoid contains negotiation information. The small geometric shapes above domain models, designs, and integrations depict **Oz** development operations (e.g., `CreateModel CreateDesign`); they record the order and application operations used in the development process.

Figure 16 summarizes the process of perspective acquisition. In this process, designers formally represent the goals and preferences of stakeholders. The desired states are represented as goal states which the automated planner understands. Model preferences are used to guide the planner when choices are available (e.g., among alternative operators). Finally, Oz allows the specification of alternative resolution methods. In this chapter, two such methods are described: interactive resolution and *a priori* resolution. However, Oz currently only supports interactive resolution. (There is a Lisp function interface for programmers who wish to apply an alternative method.)

Stakeholder perspectives contain the input required by Oz's automated planner to create designs. Additionally, these domain models contain the basis for Oz's resolution generation method. Hence, each perspective must have the same initial conditions, operators, objects, and relations; furthermore, these items must be organized in the same hierarchies. That is, the specialized domain models must be the same except for the preferences. In the face of conflict, the individual preferences over the operators, objects, and relations are used to synthesize resolution alternatives.

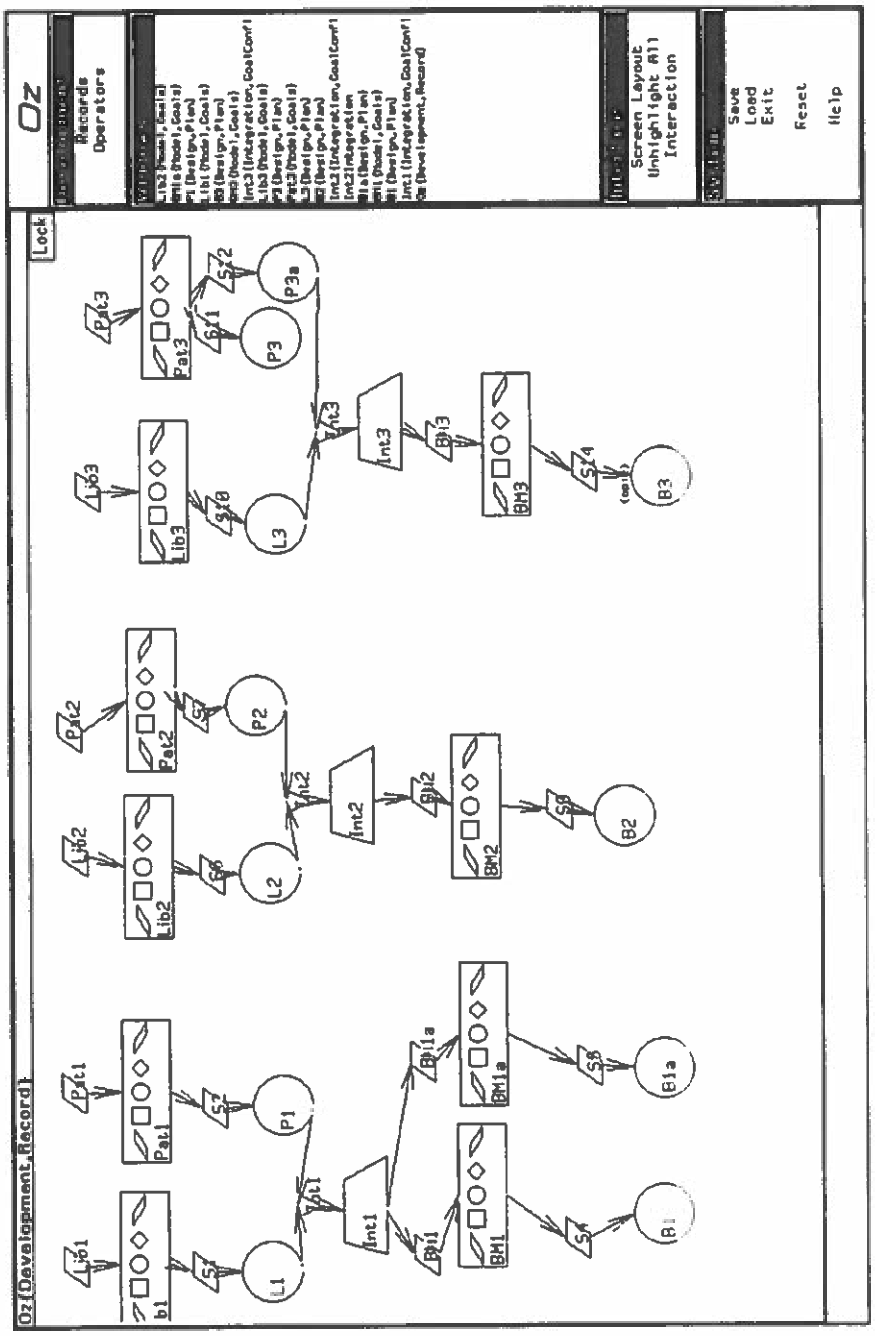
In sum, Oz's perspectives represent:

• a domain model of operators, objects, and relations

• goal states

• (optional) preferences of alternative goal states

• preferences of domain model components



Lib2(Model,Goals)
BM1a(Model,Goals)
P1(Design,Plan)
Lib1(Model,Goals)
B3(Design,Plan)
BM3(Model,Goals)
Int3(Integration,GoalConf)
Lib3(Model,Goals)
P3(Design,Plan)
Pat3(Model,Goals)
L3(Design,Plan)
B2(Design,Plan)
Int2(Integration,GoalConf)
Int2(Integration)
B1a(Design,Plan)
BM1(Model,Goals)
B1(Design,Plan)
Int1(Integration,GoalConf)
Oz(Development,Record)

Figure 15. Oz Screen Depiction of Group Design.

(1) *Describe Goal States*
Describe desired states using `def_establish` and undesirable states using `def_avoid`. Optionally, order or scale these goals indicating the preferred order of relaxation in case of conflict.

(3) *Describe Model Preferences*
Modify the domain model to reflect individual preferences concerning operators, objects, and relations. The functions, `def_mod_XX`, where XX is a operator, object, or relation, modify such preferences.

(4) *Describe Resolution Method*
Optionally, define the resolution method using `def_resolution_method`. Of the two described methods, interactive and *a priori*, only the interactive is implemented.

Figure 16. Perspective Acquisition in Oz.

• (optional) resolution method

Next, Oz perspective representations are described.

1. Model Construction

Model construction consists of representing domain operators, objects, and relations. Perspective acquisition consists of specializing the domain model by attaching goals and preferences. This section presents model construction, while the following presents perspective acquisition.

Oz model construction is greatly influenced by the the automated designer. Oz uses the abstract planner developed by Anderson and Farley, called OPIE[2]. OPIE automatically creates a plan (a.k.a. design) from a domain model and goals. The next subsection summarizes research by Anderson and Farley on the operator hierarchy construction.

1.1. Domain Operators

OPIE domain operators are consistent with STRIPS-style planners[64]. Each operator contains three lists of *add*, *delete*, and *persistence* relations. The term *persistence* is used for relation instances which must exist before, during, and after operator application. A fourth list, the *object* list, specifies the types of objects found in the other lists. Figure 16 illustrates the library borrow operator.

```
(def_operator "Borrow_Resource"
 :description "A loanable RESOURCE is LOANED to an AGENT for LOAN_PERIOD."
 :objects '(agent1 agent2 resource1 time1 time2 place1 loan_period1)
 :persistences
 '((loan_period(agent1 resource1 place1 time1 loan_period1))
   (time_is(time2)))
 :deletes '((possess(agent2 resource1 place1 time1 loan_period1)))
 :adds '((possess(agent1 resource1 place1 time2 loan_period1))
        (on_loan(agent2 agent1 resource1 place1 time2 loan_period1))))
```

Figure 17. A Library BORROW Operator.

Operators are defined using the `def_operator` Lisp function. Specializations are automatically determined by OPIE's classification algorithm. However, the objects, persistences, deletes, and adds must be defined explicitly. The syntax for the system description is:

Syntax:
```
(def_operator <name>
  :description <string>
  :objects (<object>*))
  :persistences (<relation>*)
  :deletes (<relation>*)
  :adds (<relation>*))
```

1.2. Operator Hierarchy Construction

OPIE's operators are stored in a taxonomic abstraction hierarchy. Primitive operators form the leaves, while abstract operators are formed representing common added, deleted, or persistent relations. Figure 18 illustrates the abstraction of several library operators. Such abstraction is the result of pairwise comparison and is automated[2].

From individual operator definitions, an operator generalization hierarchy is formed. It is based on the similarity of operators according to their lists of relations and objects. Operators may differ if they have: (1) identical relations names but use different objects, (2) identical objects but within different relations, (2) or have an overlap of relations and objects.

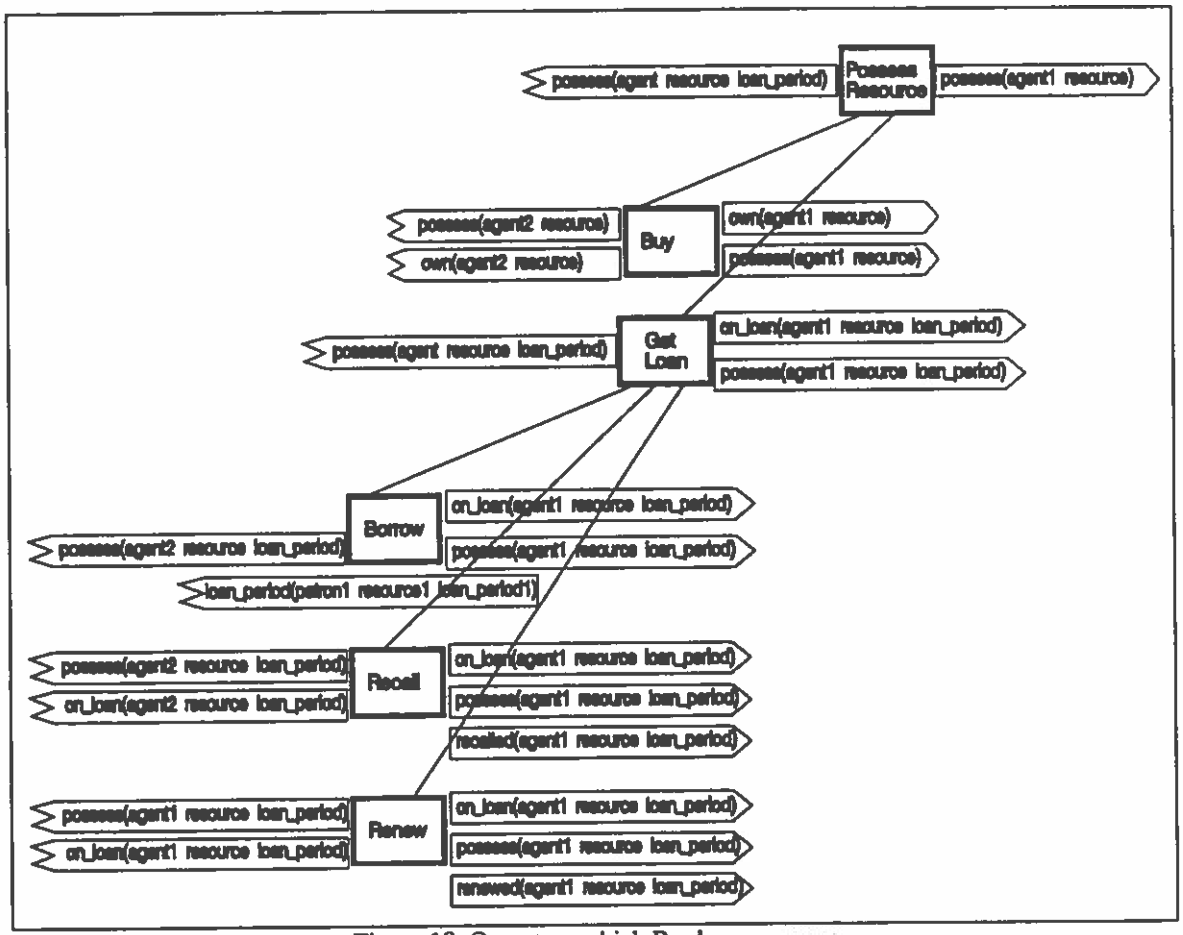

possess(agent resource loan_period)
Possess Resource
possess(agent1 resource)
possess(agent2 resource)
own(agent2 resource)
Buy
own(agent1 resource)
possess(agent1 resource)
possess(agent resource loan_period)
Get Loan
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
Borrow
on_loan(agent1 resource loan_period)
possess(agent2 resource loan_period)
possess(agent1 resource loan_period)
loan_period(patron1 resource1 loan_period1)
possess(agent2 resource loan_period)
on_loan(agent2 resource loan_period)
Recall
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
recalled(agent1 resource loan_period)
possess(agent1 resource loan_period)
on_loan(agent1 resource loan_period)
Renew
on_loan(agent1 resource loan_period)
possess(agent1 resource loan_period)
renewed(agent1 resource loan_period)

Figure 18. Operators which Produce ON_LOAN.

Figure 18 illustrates operators associated with library loaning. The three operators below GET_LOAN all have ON_LOAN in their add lists. They are also grouped by common relations on their deletes and persistences lists.

The notation of figure 18 may be intuitive. The relations on the right of an operator are *added* after operator application; relations on the left are *deleted*. Persistence relations are listed below an operator; they must exist before, during, and after an operator application.

Figure 19 presents the operator generalization algorithm. It creates a hierarchy of related operators. However, if operators do not share relations, they will be placed in a separate hierarchy. Hence, Oz's domain model contains sets of operator hierarchies. See[2] for more details.

1.3. Object Hierarchy Construction

The object hierarchy simply classifies objects into categories. For example, a RESOURCE has specializations BOOK, PERIODICAL, and SPECIAL. Similarly, GRAD, UNGRAD, and FACULTY are specializations of PATRON. Figure 20 illustrates how such categories are described.

```
For each operator NEW in the input set:
  Let set S be those leaf operators of the hierarchy which
                 share at least one add relation with NEW,
    INSERT(NEW,S).

Define INSERT(NEW,S)
  For each operator OP in S, GENERALIZE(NEW,OP).

Define GENERALIZE(NEW,OP)
  If NEW is a specialization of OP,
   Then LINK_PARENT_CHILD(OP,NEW).
   ElseIf NEW is a generalization of OP,
    Then LINK_PARENT_CHILD(NEW,OP) and
       INSERT(NEW, parent of OP).
   ElseIf NEW and OP share relations.
    Then create TMP with their common relations,
       If TMP matches existing abstract operator AB,
        Then LINK_PARENT_CHILD(AB,NEW) and discard TMP.
        Else LINK_PARENT_CHILD(TMP,NEW),
           LINK_PARENT_CHILD(TMP,OP), and
           INSERT(TMP,parents of OP).
   Else {New and OP have no shared relations}.
```
Figure 19. The Operator Generalization Algorithm.

```
(def_object "resource"
    :description "Resource types."
    :specializes '("book" "periodical" "special"))

(def_object "patron"
    :description "Patron types."
    :specializes '("grad" "ungrad" "faculty"))
```
Figure 20. The Description of Some Library Object Hierarchies.

Relationships between objects can be described via the object hierarchy. This hierarchy is used to: (1) allow the description of abstract objects, and (2) describe alternative objects for conflict resolution. When an abstract object is part of a goal, it, or any specialization of it, may be established in the design. In this way, goals containing abstract objects allow multiple acceptable concrete designs.

The object hierarchy is also used during conflict resolution. If a goal cannot be established, *generalization* attempts to achieve a related goal. One way to do this is to use the same relation, but consider alternative yet similar objects. The object hierarchy provides this information. Objects closer in the hierarchy are consider more similar than those farther apart. If a goal cannot be established, *generalization* initially considers siblings objects. In this fails, more remote objects are considered. This is one way in which substitute goals are established in the face of goal failure. Hence, the object hierarchy not only describes how relations are abstracted, but how generalization is sought.

Using the hierarchies as a basis of similarity, and hence substitution, is a common theme in the **Oz** approach. **Oz** uses the same approach to describe relations and allow for their substitution. Additionally, one could apply the same approach to operators. If operators themselves were allowed as goals (e.g., the goal is *use* BORROW) then one could use the operator hierarchy to suggest alternative operators both during individual design and during integration. Instead, **Oz** only allows the statement of goal states, thereby indirectly specifying operators. Hence, **Oz** only negotiates over goals, thereby indirectly affecting the choice of operators.

Objects are defined using the `def_object` Lisp function. Specializations are defined explicitly with the `:specializes` keyword. The syntax for system description is:

Syntax:
```
(def_object <name>
  :description <string>
  :specializes (<object>*))
```

1.4. Relation Hierarchy Construction

Relationships between relations can also be described via a hierarchy. The relation hierarchy is used to: (1) allow the description of abstract relations, and (2) describe alternative relations for conflict resolution. When an abstract goal is part of a perspective, it, or any specialization of it, may be established in the design. Abstract goals allow more operators to be applicable during design.

The relation hierarchy is also used during conflict resolution. If a goal cannot be established, *generalization* attempts attempts to established a sibling relation. In this fails, parent relations and their siblings are sought. In this way, substitute goals are established in the face of goal failure. Hence, like the object hierarchy, the relation hierarchy not only describes how relations are abstracted, but how generalization is sought.

Figure 21 illustrates two ways in which a relation hierarchy may be specified. In either case, the relation name and its objects are specified. In the case of `POSSESS`, its specializations are explicitly described. In the case of `LOAN_PERIOD`, all relations which can be made from the specialization of the objects in the object list (as defined by the object hierarchy) are formed into a hierarchy whose root is `LOAN_PERIOD(AGENT RESOURCE PLACE TIME LOAN_PERIOD)`. Figure 22 illustrates a portion of the `LOAN_PERIOD` relation hierarchy.

The significance of an explicit resolution description will become apparent during resolution generation. However, the choice of explicit, as opposed to implicit, does not effect the design

```
(def_relation "possess"
    :description "Agent physically has resource."
    :objects '("agent" "resource" "place" "time" "loan_period"))
    :specializes '(("possess"(library resource place time loan_period))
                   ("possess"(patron resource place time loan_period))))

(def_relation_all "loan_period"
    :description "A patron can borrow resource for a loan period DURATION"
    :objects '("agent" "resource" "place" "time" "loan_period"))
```
Figure 21. The Description of a Library Relation Hierarchy.

process. The relation POSSESS(LIBRARY RESOURCE PLACE TIME LOAN_PERIOD) defined in the figure 21 implicitly also allows relations using the specializations of its objects. An operator using it in a relation list can always be specialized based on the object hierarchy. However, it does effect resolution. Oz only uses the explicitly described relations during resolution generation. Hence, explicit relation description forms the basis of a simple resolution control mechanism.

Relations are defined using the `def_relation` and `def_relation_all` Lisp functions. Specializations are defined explicitly with the `:specializes` keyword or through the `def_relation_all` function. The syntax for system description is:

Syntax:
```
(def_relation <name>
  :description <string>
  :objects (<object>*))
  :specializes (<relation>*))
```

1.5. Domain Preferences

After the hierarchies are specified, they are manually modified to contain the goals and preferences of stakeholders. In annotating the hierarchies, Oz distinguishes between attributes and preferences. An ordering of items is a preference. For example, objects can be ordered. However, item themselves can be annotated with attributes. Attributes are nonfunctional annotations used to

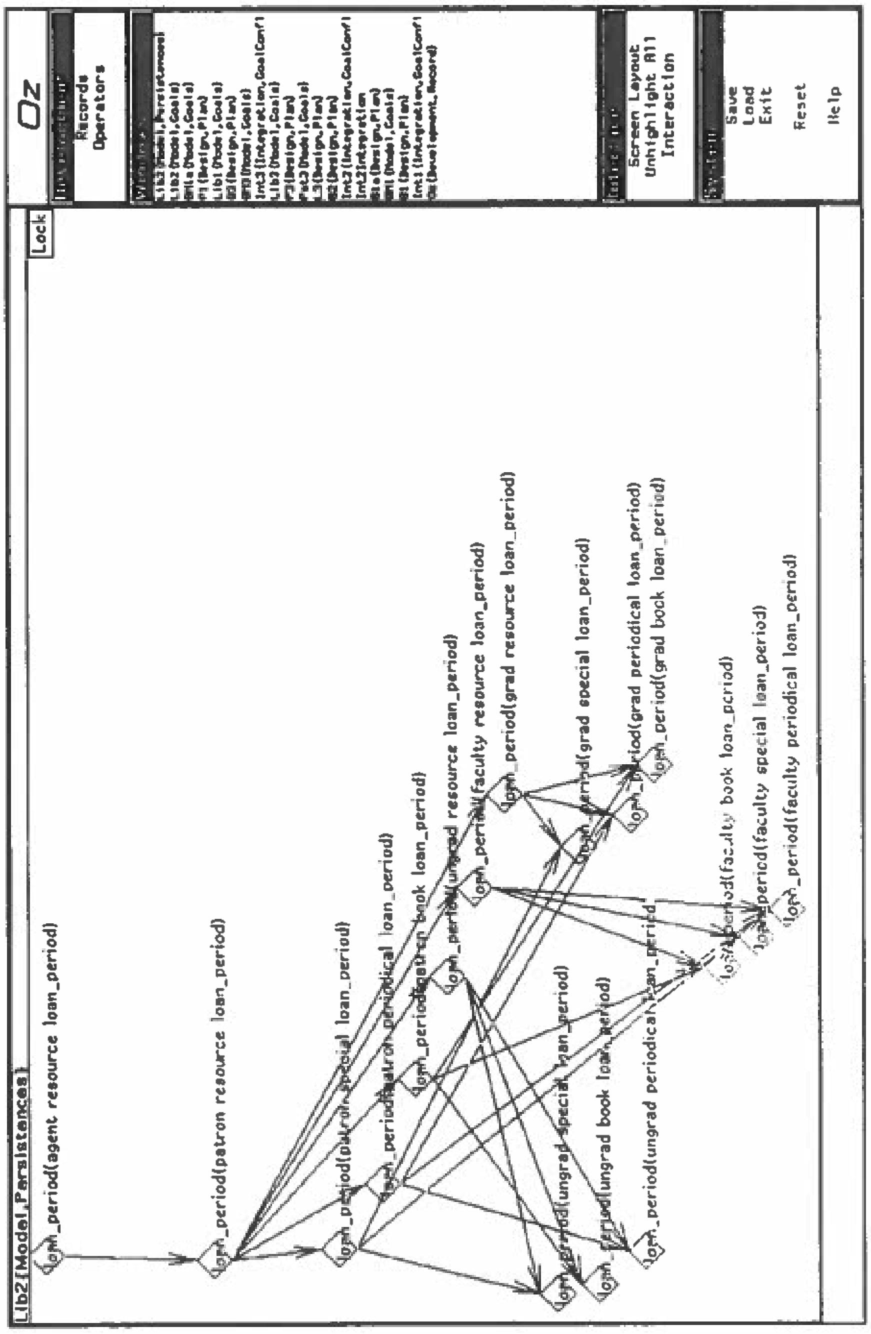

Lib2(Model,Persistences)
Lock
loan_period(agent resource loan_period)
loan_period(patron resource loan_period)
loan_period(patron special loan_period)
loan_period(patron periodical loan_period)
loan_period(patron book loan_period)
loan_period(ungrad resource loan_period)
loan_period(faculty resource loan_period)
loan_period(grad resource loan_period)
loan_period(ungrad special loan_period)
loan_period(ungrad book loan_period)
loan_period(ungrad periodical loan_period)
loan_period(grad special loan_period)
loan_period(grad periodical loan_period)
loan_period(grad book loan_period)
loan_period(faculty book loan_period)
loan_period(faculty special loan_period)
loan_period(faculty periodical loan_period)
Oz
Records
Operators
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help

Figure 22. An Oz Depiction of Library Relation Hierarchies.

differentiate items along multiple dimensions, or *criteria*. For example, operators may have different preferred orderings based on different attributes. An ordering based on availability may be the inverse of that based on cost. Next, attribute definition is described. Preference definition follows.

1.5.1. Attributes

Attribute definition has two parts: name and range, and objective. They can be defined as part of the generic model definition or as a modification of modeled operators, objects, and relations. For example, the loan period relation can be defined with an attribute in the generic model as follows:

```
(def_object "loan_period"
 :description "A period for which loans hold. Duration specifies length.")
 :attributes '(;;Ranges are the same for all models.
                    (duration :min 0 :max 365)))
```

Similarly, the duration attribute could have been defined as the modification of the loan period relation during perspective acquisition as follows:

```
(def_mod_object "loan_period"
 :description "A period for which loans hold. Duration specifies length."
 :attributes '(;;Ranges are the same for all models.
                    (duration :min 0 :max 365 :objective 14)))
```

Notice, that during perspective acquisition an attribute preference was also specified. Generally, there's no reason to define an attribute in a stakeholder's perspective unless the stakeholder has some associated preference.

Attributes (and preferences) are defined using the `def_XX` and `def_mod_XX` Lisp functions (where xx is one of operator, object, or relation). The syntax for system description for including attributes and preferences are the same as those for `def_operator`, `def_object`, and `def_relation` (and their `def_mod`'s), except for the addition of the attribute clause which is defined below.

```
Syntax:
  :attribute (<attribute-def>*)

<attribute-def> :: (<name> :min <value> :max <value> :objective <objective>)
<objective>     :: <value> | min | max
```

1.5.2. Preferences

Preferences are used to create partial orders over operators, objects, and relations. An order can be *direct*, as in the following description of ACCESS relation preferences, where ON_LOAN is preferred over OWN.

```
(def_mod_relation access(patron,resource)
  :order '(own(patron,resource)
           on_loan(patron,resource,loan_period)))
```

(Items toward the right are preferred over items toward the left.) An alternative description of this ordering is to attach an *ordering value* to each relation:

```
(def_mod_relation access(patron,resource)
  :order-by-value '((own(patron,resource)   40)
                    (on_loan(patron,resource,loan_period) 60)))
```

It is possible to describe such a preference since OWN and ON_LOAN are part of the same relation hierarchy, i.e., they are children of ACCESS. In contrast, **Oz** does not allow the *a priori* description of preferences between relations in different hierarchies. For example, assume SECURE_RECORDS and KNOW_RECORDS are in different relation hierarchies. In **Oz**, one can not state, *a priori*, that one prefers secured records over known records (if the two conflict). However, assuming plans which satisfy the two relations do conflict, then during conflict resolution one can choose which goal one wishes to achieve.

Preferences can be *implied*, as in the following description of loan period durations, where longer durations are preferred over shorter durations.

```
(def_mod_object "loan_period"
 :description "A period for which loans hold. Duration specifies length."
 :attributes '(;;Ranges are the same for all models.
                       (duration :min 0 :max 365 :objective max)))
```

In Oz, a preference is defined by an order among specializations of an item, or as an objective of an attribute. By using the `:order` keyword of instead of the `:specializes` keyword, specialized items (operators, objects, and relations) are associated with a preference of use. Alternatively, one can use attributes to associate multiple orderings of items. An attribute, like duration above, can have a specific preferred value (.e.g., 14) or a preferred direction of satisfaction (e.g., max or min). Associating maximize with loan period duration describes the preference for the largest values possible within the range of values. These are the only ways Oz currently understands preferences.

An direct expansion of Oz's description of preferences would be to allow the statement of abstract relations using maximize or minimize. For example, given the following description of resource objects and their preferred order of use, one could state a goal differently.

```
(def_object "resource"
    :description "Resource types."
    :order '("special" "book" "periodical"))
```

Instead of `ON_LOAN(PATRON,RESOURCE,LOAN_PERIOD)`, one could state `ON_LOAN(PATRON,max,LOAN_PERIOD)`, thereby implying the desire to maximize the preference for a type of resource loan. However, while Oz does not understand goals stated in this fashion, ordering resource objects has the same effect during resolution.

1.5.3. Preference Consistency in Hierarchies

The interaction of the three hierarchies can be seen in the above example. A preference in one hierarchy often implies an ordering in another. For example, the above resource ordering implies the following relation hierarchy ordering:

```
(def_relation "on_loan"
    :objects '("patron","resource","loan_period")
    :order '(("on_loan"(patron,special,loan_period))
             ("on_loan"(patron,book,loan_period))
             ("on_loan"(patron,periodical,loan_period))))
```

Similarly, relation orderings can imply operator orderings. Oz does not explicitly check or resolve inconsistencies between hierarchy orderings. Instead, as described in chapter V, it applies relation preferences and object preferences; it currently does not consider operator preferences.

1.6. Functional Goals

Functional goals describe the desired result of the design process. The planning system will attempt to find a plan which achieves the desired goals. Optionally, one can use relation preferences to describe alternative ways to "back-off" the desired goal state. (As noted in section 1.5.2, such *a priori* preferences can only be specified within a relation hierarchy.)

Functional goals are defined using the `def_establish` Lisp function. The syntax for system description is:

Syntax:
```
(def_establish (<relation>*))
```

Example:

```
(def_establish
  '((on_loan(library1 patron1 resource1 place1 time2 loan_period1))))
```

Notice, in the above example, objects have numbers appended to their names. These numbers indicate the desire to achieve the ON_LOAN relation with specific instantiations of objects. For example, PATRON1 refers to a specific patron, not the abstract class of patrons.

In addition to specifying the desire to achieve particular goal states, one can specify the desire to avoid particular states. Such states are specified similar to achievement goals. These avoidance goals are typically used in conjunction with achievement goals. They indicate to the planner that the goal states should be achieved, but without the achievement of the avoidance states.

Similar to achievement goals, avoidance goals are defined using the `def_avoid` Lisp function. The syntax for system description is:

Syntax:
```
(def_avoid (<relation>*))
```

1.7. Preference Tradeoffs

Preferences are used during design and integration to guide decisions. During design, alternative component specializations may be available. Preferences are used to choose among them. Similarly, during design some functional goals may block the achievement of others. *A priori* functional goal ordering can be used to choose among them. If all preferences do not indicate the same alternative (incompatible preferences), one could choose at random, have a human make the choice, or apply a predefined procedure to make such choices. One can apply this approach to both design and integration. However, Oz only provides support for human intervention via an interactive resolution choice aid. To contrast this with a more conventional approach, a predefined decision procedure using utility analysis is presented.

1.7.1. Interactive

Preference tradeoffs can be acquired during conflict resolution. They describe the relationship between preferences. For example, cost preferences may have a higher priority than aesthetic preferences. Rather than specify tradeoffs *a priori*, they can be acquired when they are needed, i.e., when a conflict arises. For a particular design, two preferences may never be in conflict. Hence, specification of their tradeoffs may be wasted effort; similarly, specification of the preferences themselves (the ordering) may be wasted. However, given a conflict, both preferences and their tradeoffs can be used to choose a resolution. Oz's interactive resolution method prompts for tradeoffs only during conflict resolution. And, while Oz allows the specification of preferences (i.e., the ordering of components), these too can be put off until conflict resolution.

When there are alternatives, and preferences are incomplete or incompatible, an interactive aid is used to derive tradeoffs. Oz implements Zeleny's Interactive Decision Evolution Aid (IDEA)[92]. Figure 23 illustrates how IDEA graphically represents nondominated alternatives using bar diagrams.[1] On the left is a normalized scale. A set of bars represents an alternative. The shading of a single bar represents the achievement that an alternative provides to a preference. Other possible degrees of achievement for a preference are indicated with hashed marks. A group

[1]While Zeleny outlined IDEA, its details and implementation remain unexplored.

of bars represents all the preferences of an alternative. The range of a bar represents the range of a preference. (For example, in the first preference, zero percent is not feasible, while in the second preference 100 percent is not feasible.) The higher all values are, the better the alternative. The ideal is achieved when all preferences are at the top of their range. The anti-ideal is achieved when all preference are at the bottom on their range.

Initially, all preference achievements are at zero (the anti-ideal). The decision maker moves toward the ideal by increasing the weight on particular preferences. Telling IDEA to increase achievement of the first preference causes the system to search through the alternatives and display an alternative with higher achievement of the first preference. By manipulating the preference view of the alternative set, the user can consider preferences of various alternatives. Interactions surface when increasing a preference's achievement decreases another. For example, moving from (b) to (c) in figure 24 shows the interaction between the first and second preference. Such interaction indicates that there does not exist an alternative which achieves both preferences at their highest level. Assuming a complete and accurate resolution generation procedure, this

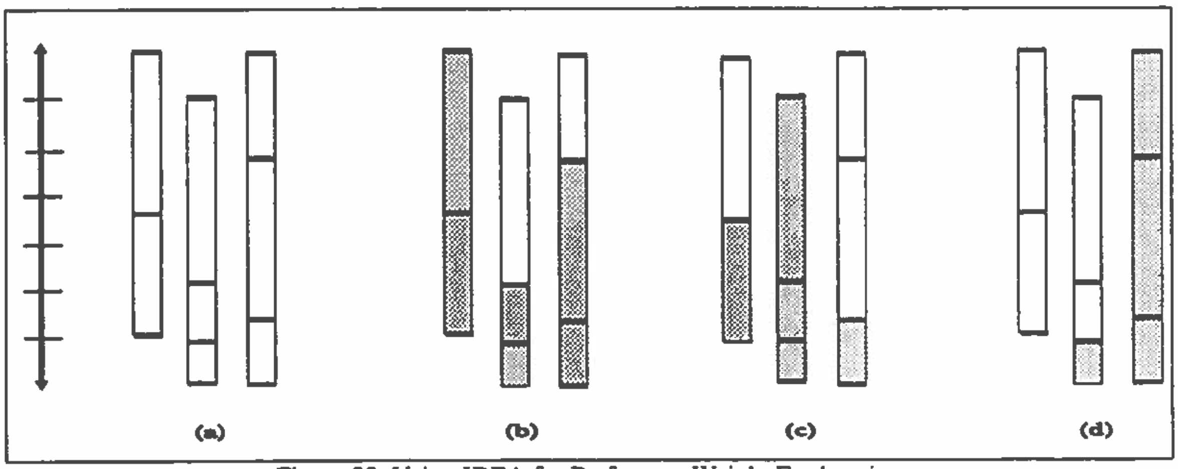


Figure 23. Using IDEA for Preference Weight Exploration.

indicates that simultaneous achievement of all preferences is not feasible. Hence, the user must decide which combination of preference achievement they want.

The Oz implementation of IDEA displays:

(1) The relation containing the conflict.

(2) The conflicts within the relation.

(3) The initial preferences for the conflict.

(4) Derived resolutions containing their relations and preferences.

Additionally, other information concerning the conflict can be had through queries to the system.

Figure 24 shows an screen dump of Oz's interactive aid. The Oz implementation differs from Zeleny's IDEA in that:

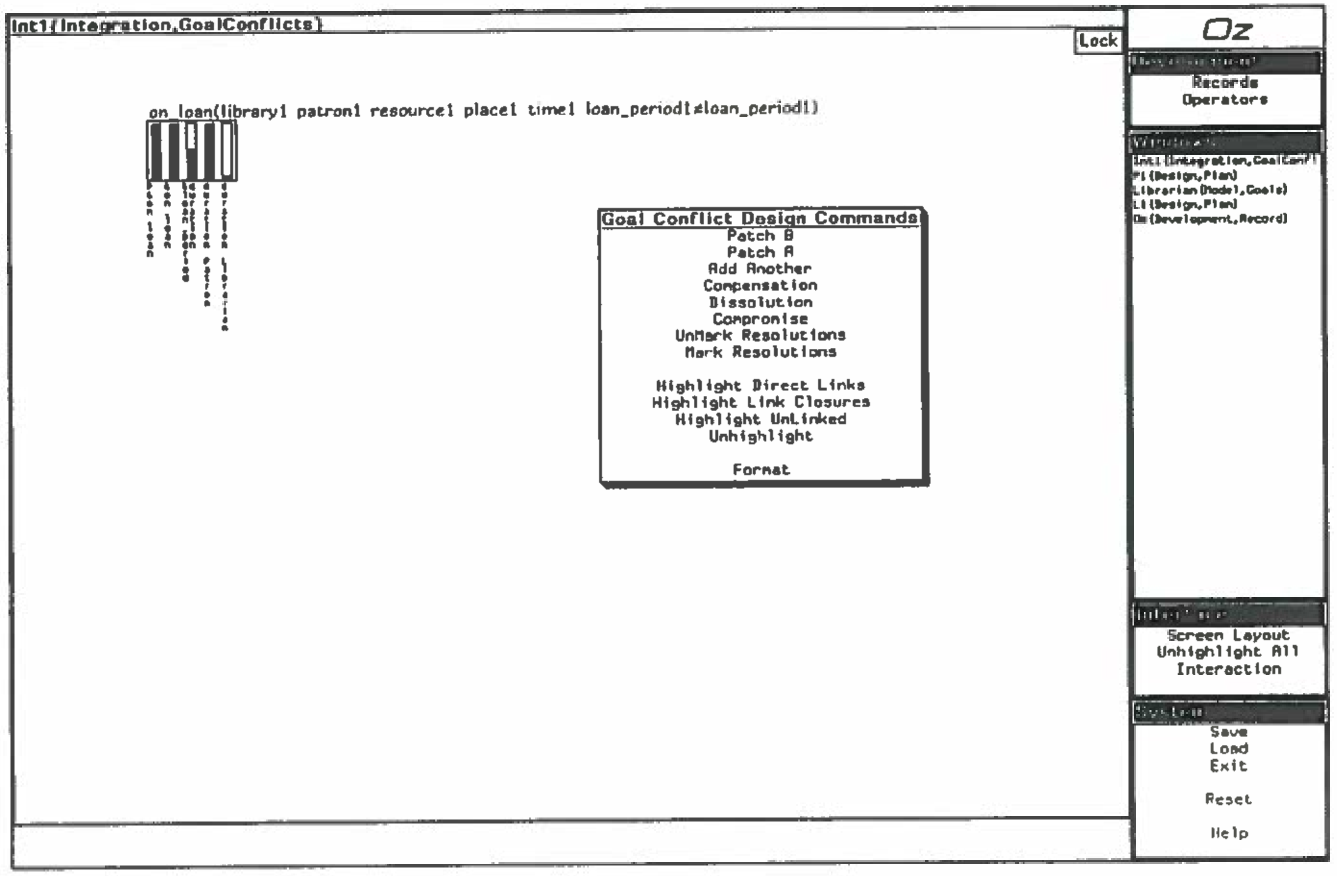
(1) Alternatives are not created through interactive increases in preferences, instead all alternatives (for a given generation method) are displayed at once.

(2) Alternative values for preferences are not hash marked on each preference bar. Instead, different preference values can be seen by viewing the displayed alternatives.

(3) Multiple preference perspectives are displayed. Additionally, their average is displayed (preceded by the "&" symbol).

Also, Oz displays the averaged preferences for: (1) the attributes of the relation (e.g., &ON_LOAN), and (2) the preferences over the relation itself (e.g., P&ON_LOAN). When there are no preferences in these two cases, the bar for that attribute is shaded at 100 percent (as shown in the figure).

When the decision maker has reached a satisfactory weighting, the associated alternative is chosen. One can consider this process in terms of utility analysis. Each preference can be



```
on_loan(library1 patron1 resource1 place1 time1 loan_period1≠loan_period1)
```

Figure 24. An Oz Depiction of IDEA.

associated with the numerical value of its percent satisfaction (e.g., 50 for 50 percent). One approach is to maximize the summation of all preferences. However, some preference may be more important than others. To indicate this, one can multiply each preference's numeric value by a weight between zero and one; preferences of less importance have less weight. Using such a weighting scheme, the "best" alternative can be recognized by its high score. One can interpret the interactive procedure in this light. In choosing an alternative, the decision maker has implicitly settled on a weighting of preferences; those preferences with less shading receive less weight. The following section describes how such a weighting scheme may be defined *a priori*. However, I believe the interactive method is superior since: (1) only necessary preferences are acquired, and (2) preferences are acquired in the context of the conflict.

1.7.2. A Priori

One can apply an additive utility model to choose a resolution. First, one needs to specify component preferences. Next, one needs to specify tradeoffs between those preferences. When a conflict occurs, one can choose a resolution by picking the resolution whose summed weighted preferences have the highest value. (In the case of ties, one is chosen at random.)

Here, it is assumed that preferences have been specified as described in the previous sections. Now, one needs to describe: (1) individual tradeoffs, and (2) perspective tradeoffs.

To describe individual tradeoffs, one needs to divide all preferences into subsets which can be simultaneous involved in a conflict. For example, a model may contain the attributes cost and risk throughout the model. Then one can globally specify the tradeoff between cost and risk, e.g., utility = 0.8 * cost + 0.2 * risk. The same can apply to component preferences. For example, assume patron specializations and resource specializations are ordered. Further, assume they both can be involved in a conflict. Then trade-offs need to specified between patron specializations and resource specializations.

For example, consider the following two (simplistic) resolutions of a loan period conflict:

Resolution-1:
```
on_loan(graduate,book,loan_period.duration=14)
```

Resolution-2:
```
on_loan(faculty,periodical,loan_period.duration=14)
```

In the first resolution, graduates receive books, while in the second faculty receive periodicals. Assume books are preferred over periodicals, while faculty are preferred over graduates. Which resolution is better? If a trade-off is specified between resources and patrons, the two categories, then a resolution can be picked automatically. For example, if the tradeoff is defined as: utility = 0.9 * patron + 0.1 resource, then the second resolution would be picked. Specifying all such tradeoffs for a large hierarchy is time consuming, tedious work.

To describe perspective tradeoffs, one can simply use global weights. For example, utility = 0.6 * perspective-1 + 0.4 * perspective-2. Alternatively, one can specify the tradeoff between every preference in the perspectives. In any case, the general form of the utility procedure to choose resolutions is:

Max $\Sigma^{p}_{n=1} \Sigma^{i}_{m=1} P_{n,m} * I_{n,m} * V_{n,m}$, where
p is the number of perspectives,
i is the number of issues (preference domains),
$P_{n,m}$ is the perspective weight associated with an issue,
$I_{n,m}$ is the individual weight associated with an issue,
$V_{n,m}$ is the value of a component.

Depending on how much one relies on global weights, specifying all numbers for this procedure can be taxing. Instead, Oz simply acquires needed weights during resolution search.

1.8. An Example

This chapter closes by presenting patron preferences used to resolve a simple loan period conflict.

```
(in-perspective 'Patron)

(def_establish
  (on_loan(patron1,resource1,loan_period1))
  (knows_records(patron1,patron1,resource1)))

(def_mod_object loan_period
  :attributes '((duration :min 0 :max 365 :goal max)))

(def_mod_relation on_loan(patron,resource,loan_period)
  :order '((on_loan(patron,special,loan_period))
           (on_loan(patron,book,loan_period))
           (on_loan(patron,periodical,loan_period)))
  :objective 'min)
```

This patron perspective states that the patron: (1) wants to achieve patron loaning and patron knowledge of individual borrowing records, (2) prefers longer loan period durations, and (3) if necessary, prefers special resources over books, and prefers books over periodicals.

2. Summary

In sum, Oz perspectives represent:

• a domain model of operators, objects, and relations

• goal states

• (optional) preferences of alternative goal states

• preferences of domain model components

• (optional) resolution method

Using such perspectives, the Oz planner can automatically create designs and the resolution methods can automatically generate resolutions.

CHAPTER IV

AUTOMATED DESIGN

The purpose of design is to show that, given the operator set, it is possible to derive a design which achieves a goal. For us, design is a sufficiency consideration. Is the current domain model sufficient to satisfy the goals of each stakeholder's perspectives? If the answer is yes, then is the current domain model sufficient to satisfy the goals of all stakeholder's perspectives simultaneously? The integration process determines this by checking for design interference. If there is interference, Oz first seeks alternative (sub)designs via replanning. If the perspectives inherently lead to conflicting designs, Oz aids the negotiation of a group perspective free of conflict.

This chapter presents Anderson and Farley's automated planner. Since the design process itself is not the main focus, this presentation is brief. First, the design process is presented by way of an illustrative example. Next, *incorporation links* are described. This part of this design record is required by the automated conflict detection method (described in chapter V).

1. A Perspective

Below is a simple perspective of a library patron.

```
(in-perspective 'Patron)

(def_establish
  '(renewed(agent1 resource1 loan_period1)))
```

The patron has the goal of renewing a resource. To make the example slightly more interesting, assume that the patron has not yet borrowed any resources. (Perhaps, an analyst wants to test the results of a pathological case.)

2. Initial State Description

Below is the initial state given to the planner.

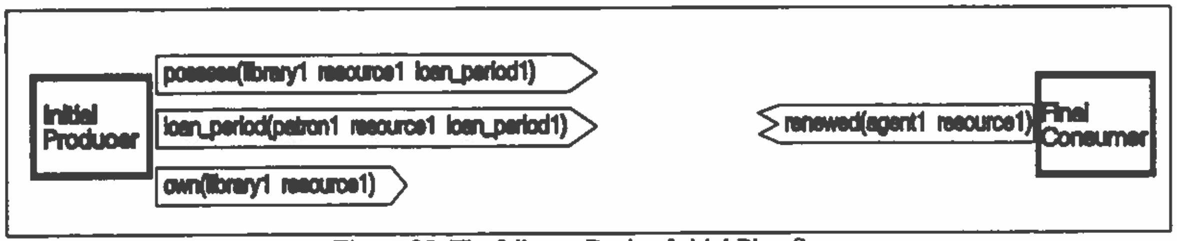
```
(in-perspective 'Library-Model)

(def_initial_state
  '((possess(library1 resource1 loan_period1))
      (own(library1 resource1))
      (loan_period(patron1 resource1 loan_period1))))
```

It describes the initial state of the library system as having: (1) one resource which is possessed and owned by the library, and (2) one patron loan period. The number appended to the object categories simply indicate that these are specific instances of those objects.

This initial state is represented in the OPIE planning system as being *added* by an initial operator, called the INITIAL PRODUCER. The goal from the patron's perspective will be represented similarly. A operator, called the FINAL CONSUMER is placed in the system; it will delete the goal RENEWED(AGENT1 RESOURCE1 LOAN_PERIOD1). The task of planner is to insert and connect operators between the initial producer and the final consumer. Such operators constitute a plan to achieve the goal given the current state of the library.

Figure 25 illustrates the initial and final goal states. On the left, the INITIAL_PRODUCER operator produces the three initial relation instances. On the right is the single goal state desired by the patron. Anderson and Farley describe how their abstract *partial commitment* planner, OPIE, can efficiently produce plans from such input[2]. Their algorithm, slightly modified, is reproduced in figure 26. The first modification is to change the success criteria (2b). OPIE must produce a plan containing only concrete (leaf) operators. I have modified OPIE to allow the use


Initial Producer
possess(library1 resource1 loan_period1)
loan_period(patron1 resource1 loan_period1)
own(library1 resource1)
renewed(agent1 resource1)
Final Consumer
Figure 25. The Library Design Initial Plan State.

of abstract operators. If an abstract operator achieves the required relations within a plan, it will not be refined. This reduces conflicts over objects which stakeholders have no preference; that is, what they consider implementation details. Additionally, I have modified the OPIE algorithm to take advantage of preferences when there is a choice among component specializations (2c). Rather than arbitrarily work on alternative subplans, the modified OPIE (will) work on the most preferred plans first. I say will, because only modification 2b is currently implemented. Modification 2b was chosen because it is critical to the resolution process, while 2c is more of a design concern (albeit possibility effecting resolution).

3. Partial Commitment Planning

(1) Create a node containing the initial producer and final consumer and place it in the search queue.

(2) Select a node from the queue:
(a) if the search queue is empty, fail;
(b) if the node satisfies the success criteria, report success and return the node;
(c) choose the node based on the preferences.

(3) Refine the node:
(a) generate a child node for each possible refinement;
(b) complete each new node by propagating constraints.

(4) Evaluate each new node:
(a) if any constraint is violated, reject the node;
(b) else, add the node to the search queue.

(5) Go to step 2.

Figure 26. Partial Commitment Planning in Oz's OPIE.

Anderson and Farley's abstract planner, called OPIE, searches plan states. Operators are inserted into the plan based on the relations they *add* and *delete*. For example, figure 27 illustrates how RENEW RESOURCE can be inserted to add the final goal instantiation. RENEW RESOURCE's added relations are depicted to the right; they will be deleted by the final operator (FINAL_CONSUMER). RENEW RESOURCE's required relations are depicted to the left; they must be added earlier in the plan. Finally, as stated in chapter I (§3.3.2), there may be persistent relations which must be true before, during, and after the execution of an operator. (RENEW has no persistent relations.)

OPIE uses a simple best first search strategy augmented with heuristics to guide search. In addition, OPIE's operator abstract hierarchy is used to reduce search complexity[2]. Abstract operators are inserted into plans where two (or more) operators may apply; hence, a commitment is made to a subset of operators. As planning continues, constraints may naturally eliminate some operators. Unification of an operator's add list against delete lists determines if an operator is applicable.

Figure 28 illustrates the hierarchy from which the RENEW and BORROW operators were drawn. OPIE automates design in the following way:

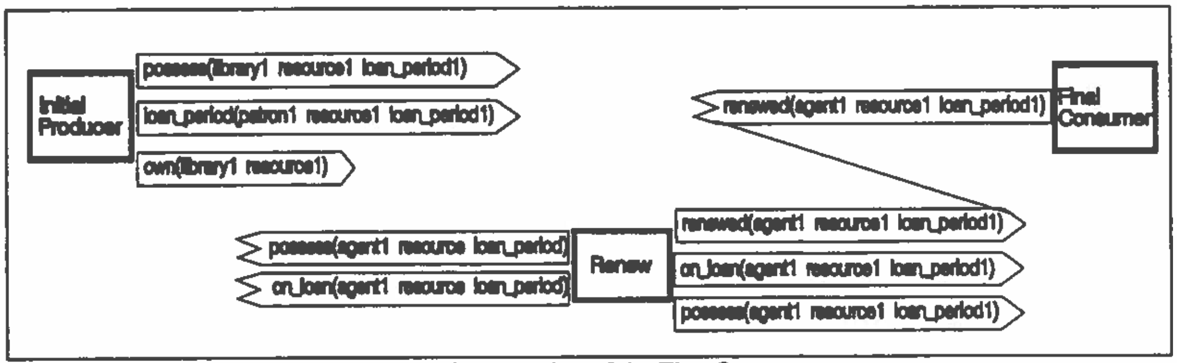

Initial Producer
possess(library1 resource1 loan_period1)
loan_period(patron1 resource1 loan_period1)
own(library1 resource1)
renewed(agent1 resource1 loan_period1)
Final Consumer
possess(agent1 resource loan_period)
on_loan(agent1 resource loan_period)
Renew
renewed(agent1 resource1 loan_period1)
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1 loan_period1)

Figure 27. Insertion of the First Operator.

(1) The most abstract operator (RENEW) for achieving a goal (RENEWED(AGENT1 RESOURCE1 LOAN_PERIOD1)) is found.

(2) If there is a preference among the children of O, the most preferred child is chosen next, and step 2 is repeated with the child becoming O. In this example, there are no operator preferences.

(3) If no preference is given among the children of an operator, or if an operator has no children, then it is selected. Hence, OPIE only specializes when necessary, otherwise leaving designs in their abstract state.

In this same manner, the rest of the design is completed. Next, the delete (goal) relations of RENEW become the focus of OPIE's attention. An operator is found (BORROW) and inserted into the plan. When finally there are no ununified relation instances, the planner is successful.

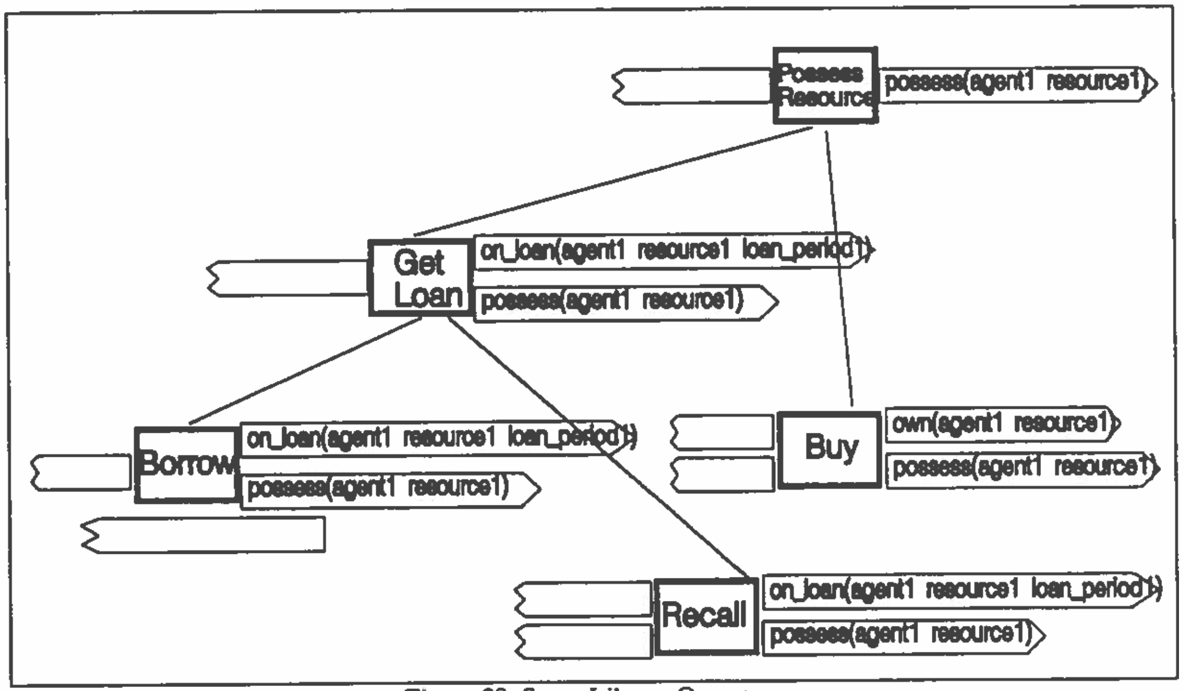
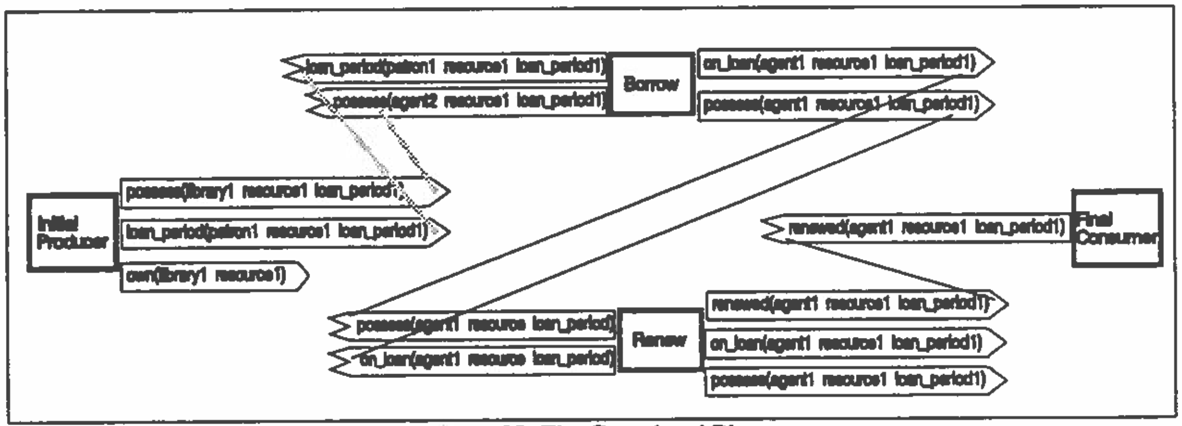

Possess Resource
possess(agent1 resource1)
Get Loan
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1)
Borrow
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1)
Buy
own(agent1 resource1)
possess(agent1 resource1)
Recall
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1)

Figure 28. Some Library Operators.

Design is completed when all goals are established. Figure 29 illustrates the completed renewal plan. The two operators, BORROW and RENEW, are illustrated with their relation unifications (depicted as shaded links). In this particular case, the patron's goal created a plan with only concrete operators. However, as shown in chapter I (§3.3.3) plans can be created which contain abstract operators.

4. Incorporation Links

After a design is completed, it will be necessary to understand its derivation. To aid this, Oz keep incorporation links during plan refinement. An incorporation link ($\rightarrow$) is placed in the design record when a goal, G, requires the establishment of an abstract component, C^0, which is then refined through a series of components (c^1, c^2, ... c^n) to the plan component, c^{n+1}.


loan_period(patron1 resource1 loan_period1)
possess(agent2 resource1 loan_period1)
Borrow
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1 loan_period1)
possess(library1 resource1 loan_period1)
Initial Producer
loan_period(patron1 resource1 loan_period1)
own(library1 resource1)
renewed(agent1 resource1 loan_period1)
Final Consumer
possess(agent1 resource loan_period)
on_loan(agent1 resource loan_period)
Renew
renewed(agent1 resource1 loan_period1)
on_loan(agent1 resource1 loan_period1)
possess(agent1 resource1 loan_period1)
Figure 29. The Completed Plan.

Incorporation Link (initial)
$G \rightarrow C^0 \rightarrow c^1 \rightarrow \ldots c^{n+1} \Rightarrow$
$\mathit{FUNCTIONAL\_GOAL}(G)$ ☞ $\mathit{ESTABLISHED}(C^0) \wedge$
$\mathit{ESTABLISHED}(C^0)$ ☞ $\mathit{ESTABLISHED}(c_j^1) \mid c_j^1 \leftrightarrow C^0 \wedge$
...
$\mathit{ESTABLISHED}(c_k^n)$ ☞ $\mathit{ESTABLISHED}(c_l^{n+1}) \mid c_k^n \leftrightarrow c_l^{n+1}$

where θ_i ☞ θ_j *means that given* θ_i *the system inferred and produced* θ_j.

Incorporation links allow one to trace from a plan component, c, to the functional goal, G, from which it was derived. Without such links, the multiple inheritance of the hierarchy would make tracing more difficult.

The above description of incorporation links allows the trace from a goal, G, through abstract operator refinement, to a primitive operator, c. However, typically the establishment of a goal requires multiple operators at each level of abstraction. So, goals are linked to sets of operators rather than a single operator.

Incorporation Link (final)
$G \rightarrow \{C^0\ldots\} \rightarrow \{c^1\ldots\} \rightarrow \ldots \{c^{n+1}\ldots\} \Rightarrow$
$\mathit{FUNCTIONAL\_GOAL}(G)$ ☞ $\mathit{ESTABLISHED}(\{C^0\ldots\}) \wedge$
$\mathit{ESTABLISHED}(\{C^0\ldots\})$ ☞ $\mathit{ESTABLISHED}(\{c_j^1\ldots\}) \mid \{c_j^1\ldots\} \leftrightarrow \{C^0\ldots\} \wedge$
...
$\mathit{ESTABLISHED}(\{c_k^n\ldots\})$ ☞ $\mathit{ESTABLISHED}(\{c_l^{n+1}\ldots\}) \mid \{c_k^n\ldots\} \leftrightarrow \{c_l^{n+1}\ldots\}$

where θ_i ☞ θ_j *means that given* θ_i *the system inferred and produced* θ_j.

At the most abstract level, a component C_i^0 is linked to a goal G if:

- C_i^0 produces a relation which unifies with G, or
- C_i^0 produces a relation which is deleted (consumed) by C_j^0, where C_j^0 is linked to goal G.

All components at abstraction level i linked to goal G form an incorporation link set.

Incorporation links can represent function sharing. In planning for goal G_1, goal G_2 may be opportunistically established. Even when operators $c_i...c_j$ incorporated to establish G_1 coincidentally establish G_2, incorporation links are formed linking the components to both goals.

5. A Notational Aside

This section introduces a compact notation for describing the preference hierarchies described in the previous chapter. Combined with the notion of incorporation links, this notation is used to expand on the relationship between the hierarchies and the automated planner.

5.1. Component Set Notation

Let C be a set of components (operators, objects, or relations) at abstraction level i, $C^i = \{c^i_1, c^i_2, ..., c^i_n\}$. ($C$'s will be issues; they will define a domains of conflict.) Π, the hierarchies, is the set of all C's; it defines how components may be related to each other via abstraction. Members of C^i, c_j, are linked to their parent(s) ($c^i_j \leftrightarrows C^i$). A parent, in turn, may be part of a another component set, $C^{i-1} = \{C^i, ...\}$. Hence, one can follow links from the root to the leaves of the hierarchies.

5.2. Basic Constraints

Component sets can be constrained. Given a component set C, let $c_i \in C$, then

Elimination $C - \{c_1, c_2, ..., c_n\}$

constrains the set by eliminating $c_1 ... c_n$ from C. Since the hierarchy itself remains unchanged, it has the effect of dividing a set into a partial order: $C - \{c_1, c_2, ..., c_n\}$ is preferred over $\{c_1, c_2, ...,$

$c_n\}$. If members of C are numerical ($c \in C$ / $c \in \Re$), C may be numerically constrained:

Numeric Elimination $C - \{c: k \leq c \leq l\}$

The application of a set constraints is written:

$\phi_i \backslash \{\phi_1, ..., \phi_n\}$
where '\' means $\phi_i - \cup_{j=1}^{n} \phi_j$; $\phi_j \in \Phi$, the universe of sets.

Finally, component sets can be ordered arbitrarily.

Arbitrary Order $C = (c_1, c_2, ... c_k) / \forall\ c_i\ c_j \in C, p(c_i,c_j)$, *where p is a lisp predicate.*

In fact, Oz relies on the notations of the previous chapter to derive all orders and eliminations. Again, this set notation is simply a means of conceptualizing preference descriptions.

Applying constraints to the component sets annotates the hierarchy Π. If ϕ is the set of all constraints then, $\Pi \backslash \phi$ denotes the remainder of Π after constraint application. Also, as an abbreviation, $C\backslash$ refers the component set, C, after all constraints have been applied.

5.3. Ordering

One can use the set notation to describe orders. Let f be a function which orders the members of a set and $c_i \in C$ then:

Ordering $f(S) = (c_1, c_2, ..., c_n)$

Sets can also be partially ordered. Let f be a function which partially orders the members of a set and $\psi_i \in C$, or $\psi_i = [c_1, c_2, ... c_j]$ where $c_1 ... c_j \in C$, (The brackets ([]) denote an equivalence class in a partial order.) then:

Partial Ordering $f(S) = (\psi_1, \psi_2, \ldots, \psi_n)$

All orderings will be considered preference orderings. Define P as a preference function; it derives a partial order:

$$P(C) = (\psi_1 < \psi_2 < \ldots < \psi_n)$$

where a < b, means b is preferred or indifferent to a.

5.4. Scaling

Component sets can be mapped onto a range. Let m be an onto mapping function $m(\phi_i) = \{(\phi_i, r_i)\}$.

Mapping $m(C) = \{(\phi_i, r_i)\}$, *where* $r_i \in \Re$

Define S as a scaling function which maps issues onto the range $[s_{low}..s_{high}]$, e.g.:

$$S(C^i_j) = \{(c_1,s_1), (c_2,s_2), \ldots, (c_n,s_n)\} \text{ where } s_i \in [s_{low}..s_{high}]$$
$$S(C^i_j) = \{(c_1,0), (c_2,5), \ldots, (c_n,100)\} \text{ where } s_i \in [0..100]$$

For example, let C^0_0 define relations, where $C^0_0 = \{\text{OWN}, \text{ON\_LOAN}\}$.

$$S(C^0_0) = S^{0,0} = \{(\emptyset,0), (\{\text{OWN}\},40), (\{\text{ON\_LOAN}\},60)\}$$

5.5. Relation Constraints

Relation constraints are arbitrary mappings used to constrain relationships, $R(\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,k}) = (\phi_{j,1}, \phi_{j,2}, \ldots, \phi_{j,k})$. Relations constraints are defined in the following form:

$$NAME(\phi_{i,1}, \ldots, \phi_{i,k}) \Rightarrow (\phi_{j,1}, \ldots, \phi_{j,k}) \text{ where } \theta$$

where ϕ's are sets and θ is a logical formula when applied to ϕ_i results in the sets ϕ_j, $\forall k\ \phi_{i,k} \supseteq \phi_{j,k}$. Hence, any relation constraint can be reduced to constraint form:

$$\begin{aligned} NAME(\phi_{i,1}, \ldots, \phi_{i,k}) &\Rightarrow (\phi_{j,1}, \ldots, \phi_{j,k}) \text{ where } \theta \\ &\Rightarrow (\phi_{i,1} \backslash \phi_{l,1}, \ldots, \phi_{i,k} \backslash \phi_{l,k}) = (\phi_{j,1}, \ldots, \phi_{j,k}) \end{aligned}$$

The second set of arguments can be left off if they are the same as the first, e.g.:

$$\begin{aligned} NAME(\phi_{i,1}, \ldots, \phi_{i,k}) &\Rightarrow (\phi_{i,1}, \ldots, \phi_{i,k}) \text{ where } \theta \\ &= NAME(\phi_{i,1}, \ldots, \phi_{i,k}) \text{ where } \theta \end{aligned}$$

In this case, NAME expresses the constraining of ϕ, rather than the relation between two sets; such relations constrain the hierarchy Π.

5.6. Component Goals

Let's consider the establishment of a component. A component is a operator, object, or relation. (Currently, Oz only allows the description of relations as goals; however, the following holds even when objects and operators can be directly specified for achievement.) A component goal, c_j, is simply a member of a set, $c_j \in C$. There are two types of component goals, establish and avoid:

Establish Component Goal
$ESTABLISH(c_j) \Rightarrow \{c_1, \underline{c}_j, \ldots, c_n\} = ([c_1, \ldots, c_n], c_j)$
Avoid Component Goal
$AVOID(c_j) \Rightarrow \{c_1, \bar{c}_j, \ldots, c_n\} = \{c_1, \bar{c}_j, \ldots, c_n\} \backslash \{c_j\}$

Goal establishment is annotated (e.g., ESTABLISH(c)), as $\underline{c}$, and goal avoidance (e.g., AVOID(c)) as $\bar{c}$.

The intent to establish only a specific goal means that all related goal formulations are lesser preferred: $([c_1, \ldots, c_n], c_j)$. In this formulation, lesser goals may still be obtained. (Replanning after goal achievement failure is controlled by the resolution algorithms; hence, goal achievement is never all or nothing.) AVOID relations can be represented as constraints: $C \backslash \{c_j\}$; hence, the goal seeking of $C = \{\underline{c}_j\}$ can be rewritten as avoiding of the complement: $C \backslash \{C - c_j\}$.

Goals are established if they are represented in the plan, i.e., IN_PLAN(c). Goals are avoided if they are not present in the plan, i.e., ¬ IN_PLAN(c). The following ESTABLISH and AVOID mappings map goals onto $\{\varnothing, \phi\}$, where $\varnothing$ indicates the falsehood of the mapping name and ϕ indicates the set of components which fulfill the mapping name. IN_PLAN(ϕ_i) returns ϕ_j if ϕ_i is achieved in the plan using the more specific components ϕ_j.

Established Goal
$ESTABLISHED(c_j^i) \Rightarrow$
$$IN\_PLAN(c_j^i) \vee$$
$$\exists\, c_k^{i+1} : c_j^i \leftrightarrow c_k^{i+1} \wedge \exists\, ESTABLISHED(c_k^{i+1})$$
Avoided Goal
$AVOIDED(c_j^i) \Rightarrow \neg ESTABLISHED(c_k^i)$

The link between levels of abstraction $(c_j^i \leftrightarrow c_k^{i+1})$ indicates the abstraction of components, (e.g., c_j^i is specialized into c_k^{i+1}). One can modify the above relations to require all establishments (plans) to be at the lowest level:

Established Lowest Goal

$ESTABLISHED\_LOWEST(c_j^i) \Rightarrow$
$$(IN\_PLAN(c_j^i) \wedge \neg\exists\, c^{i+1} : c_j^i \leftrightarrow c^{i+1}) \vee$$
$$\exists\, c_k^{i+1} : c_j^i \leftrightarrow c_k^{i+1} \wedge \exists\, ESTABLISHED\_LOWEST(c_k^{i+1})$$

Avoided Lowest Goal

$AVOIDED\_LOWEST(c_j^i) \Rightarrow \neg ESTABLISHED\_LOWEST(c_j^i)$

Similarly, one can require that all establishments be to some level, *l*:

Established Level Goal

$l\_ESTABLISHED(c_j^i) \Rightarrow$
$$(IN\_PLAN(c_j^i) \wedge i = l) \vee$$
$$\exists\, c_k^{i+1} : c_j^i \leftrightarrow c_k^{i+1} \wedge \exists\, l\_ESTABLISHED(c_k^{i+1})$$

l_Avoided Goal

$l\_AVOIDED(c_j^i) \Rightarrow \neg l\_ESTABLISHED(c_j^i)$

Such propositions are part of the design process and not normally associated with the domain model.

5.7. Policies and Objectives

A preference simply indicates that desire to establish components according to an order; it specifies how a goal may be partially satisfied. Let c^i denote a goal at abstraction level *i*. c^i will be fully satisfied if c_j^{i+1} is established, where $\neg\exists\, c_k^{i+1} \wedge c_j < c_k$.

Max Established Goal

$MAX\_ESTABLISHED(c^i_j) \Rightarrow$

$$(IN\_PLAN(c^i_j) \wedge \neg\exists\, c^i_k: c_j \prec c_k \vee$$
$$\exists\, c^{i+1}_k: c^i_j \leftrightarrow c^{i+1}_k \wedge \neg\exists\, c^{i+1}_l / c^{i+1}_k \prec c^{i+1}_l \wedge \exists\, MAX\_ESTABLISHED(c^{i+1}_k)$$

Min Established Goal

$MIN\_ESTABLISHED(c^i_j) \Rightarrow$

$$(IN\_PLAN(c^i_j) \wedge \neg\exists\, c^{i+1}_k: c_j \succ c_k \vee$$
$$\exists\, c^{i+1}_k: c^i_j \leftrightarrow c^{i+1}_k \wedge \neg\exists\, c^{i+1}_l / c^{i+1}_k \prec c^{i+1}_l \wedge \exists\, MIN\_ESTABLISHED(c^{i+1}_k)$$

Policies are global preferences. They describe the desired to maximize or minimize components according to orders. For example, let members of $P(C)$, ψ_i, represent costs; $\psi_i \in C$, or classes of equivalent costs, $\psi_i = [c_{i,1}, c_{i,2}, ..., c_{i,n}]$. The policy of minimizing costs can be described as:

$$min(\text{COST}) = (\psi_1, \psi_2, ..., \psi_n), \text{ where } \psi_i \geq \psi_{i+1}$$
$$\text{if } \psi_i = [c_{i,1}...c_{i,n}] \text{ then } \Sigma\, \psi_i = \Sigma\, c_{i,1}$$

That is, smaller costs are preferred. Such policies can be specified in Oz. The above cost preference is:

```
(def_mod_operator_all "ALL_ROOTS"
 :attributes '((cost :objective min)))

(def_mod_object_all "ALL_ROOTS"
 :attributes '((cost :objective min)))

(def_mod_relation_all "ALL_ROOTS"
 :attributes '((cost :objective min)))
```

`ALL_ROOTS` is a reserved component name to indicate that the function should be applied to all roots. The cost minimization object will be attached to all system components. After these functions are applied, one can override this global policy with invocations of `def_mod_XX` for specific abstractions.

6. Summary

In sum, the design process consists of:

• hierarchical search for operators producing required predicates

• selection of operators

• refinement of operators

Such partial commitment planning relies on the relations and constraints contained in the operator descriptions to guide search; preferences are only applied when multiple alternatives are applicable.

CHAPTER V

INTEGRATING DESIGNS

This chapter presents the integration algorithms. First, goal conflicts are detected. Next, their design level interference is determined. All goals with interference must be resolved before an integrated design can be derived. To do so, the negotiation algorithms generate possible resolutions from which an user interactively chooses for inclusion in the final design. After the user determines the resolutions, the original goals must be transformed before being passed onto the automated planner. Finally, the planner derives a design.

In this chapter the algorithms are applied to two perspectives; however, the algorithms are easily scaled to *n* perspectives. Unfortunately, Oz does not do *n*-way integrations. As an alternative, one can applied cascaded integration to integrate more than two perspectives.

1. Conflict Detection

Conflict detection attempts to identify like concepts occurring in multiple designs. Concepts which differ will be considered conflicts. Generally, this is an instantiation of the concept recognition problem: minor variations in instantiations must be recognized as being the same concept. Rather than tackle this problem, assumptions have been applied to narrow the problem. One can rely on a human to derive correspondences. (Even in this case some automated support has been supplied.) However, since human correspondence detection is error prone, Oz uses an automated approach. Derivation records are used to identify functionally corresponding components.

This section introduces the problem with the interactive detection method. Next, an automated conflict detection is presented. Finally, the more limited algorithm, actually used in Oz, is presented.

1.1. Interactive Correspondence Identification

Previously, Oz automatically generated an initial correspondence structure and then allowed a user to edit it. Using the assumptions that correspondences can be created based on identical names and types, correspondences were created. Any perceived errors in this structure could then be edited. The two assumptions were:

Names Identical Assumption
Components c_1 and c_2 will be considered for correspondence if c_1.NAME = c_2.NAME, where c.NAME is a string and "=" is a string comparator.

Types Identical Assumption
Components c_1 and c_2 will be considered for correspondence if c_1.TYPE = c_2.TYPE, where c.TYPE $\in$ {OBJECT OPERATOR RELATION}.

Also, Oz allowed the user to define equivalence functions based on component attributes.

User Tags Identical Assumption
Components c_1 and c_2 will be considered for correspondence if c_1.USER_TAG = c_2.USER_TAG, where c.USER_TAG is a lisp value and "=" is the lisp `equal` function.

These three assumptions were included in the correspondence identification algorithm. (A user checked the desired assumptions in a dialogue box before integration.)

Below is a description of the interactive correspondence algorithm. Notice that components in design one will be linked to the first like component in design two. This is an intended side-effect of looping. This *First Component Match* heuristic was a simple consideration of the overall structure of the design.

```
create_eq_structure(design1,design2, &optional (name t) (type t) (u_tag t)) ::
 let components_1 = design1.components
 let components_2 = design2.components
 for component1 in components_1
  for component2 in components_2
   if and((if name (= component1.name component2.name) t),          ;Names are equivalent.
          (if type (= component1.type component2.type) t),          ;Types are equivalent.
          (if u_tag (= component1.u_tag component2.u_tag) t))       ;User tags are equivalent.
   then add_to(component1 ≡ component2), eq_structure)
      remove_from(component2,components_2)                          ;Don't link twice.
   else add_to(singles,component1,component2)
return (eq_structure,singles)
```

Of course, such a simple algorithm will create erroneous correspondences. The algorithm below allowed the user to edit the correspondence structure.

```
user_edit(eq_structure) ::
 loop for mouse_click = get_mouse()
  case: mouse_click = delete_correspondence
        remove(correspondence, eq_structure)
        add(correspondence.component1, singles)
        add(correspondence.component2, singles)
  case: mouse_click = add_correspondence
        add(correspondence, eq_structure)
        remove(component1, singles)
        remove(component2, singles)
 until mouse_click = exit
return eq_structure
```

The use of the automated planner has reduced the effectiveness of this interactive approach. On the other hand, it has simplified the automatic detection of conflicts. Because the planner automatically derives a set of components from a goal, its record of derivation can be used to determine correspondence. Once goals can be found to correspondence, it can be inferred that the derived components correspondence. Since Oz uses an automated planner, it can simply determine if two goals correspond based on their place in the relation hierarchy. However, since many relations look quite similar, this has been an error prone for humans. So, Oz no uses a simpler, yet fully automated, detection method.

1.2. Automatic Correspondence Identification (Complete)

Rather than rely on the intuition of a user, **Oz** can apply an automatic correspondence algorithm. It is based on a simple assumption:

> **Derived Components Correspond Assumption**
> Components c_1 and c_2 will be considered corresponding if they were derived from a common functional goal, G.

A correspondence structure will be built if: (1) two goals, G_1 and G_2, are held by two different agents and (2) the goals are of the same type, and (3) the goal are the best match compared to the other goals in the designs. The closeness of match is determined by the attributes associated with goals. If goals G_1 and G_2 are of the same type, but have different attributes, and a third goal, G_3 is of the same type and has the same attributes as G_1, then G_1 and G_3 will be paired.

The actual conflict detection algorithm simply marks goals and their derived components as corresponding. A more robust algorithm would derive a hierarchical correspondence structure. The structure would consists of the goals, a common abstract operator, possibly shared derivation, and possibly a derivation divergence.

To create a hierarchical correspondence structure, first the common root is identified. Next, operator refinements (i.e., incorporation links) are traced down the hierarchy. Where the derivation is identical, no correspondence structure is needed. Where the derivation diverges, the structure notes correspondence between operators at each abstract level, i. Eventually, different sets of primitive operators are said to correspond.

Figure 30 presents a hierarchical correspondence algorithm. First, `derivation_difference` determines where the two derivations diverge. Next, `derived_correspondences` constructs a hierarchical correspondence list.

1.3. Automatic Correspondence Detection (Actual)

To date, I have not found the hierarchical correspondence algorithm necessary. Instead, Oz uses a simpler method. First, goals are matched by their most specific conflict type. This determines goal correspondence. Next, their derived components are found. This determines design level correspondence. Finally, these correspondences are passed onto the interference algorithm described in the next section.

Figure 31 shows the goal conflict detection algorithm. `Find-conflicts` first gathers all

```
derivation_difference(goal1,goal2) ::
 if =(derived_op(goal1),derived_op(goal2)) then
   return derivation_diff(derived_op(goal1),derived_op(goal2),goal1)
 else return t                                   ;No common derivation.
(defun derivation_diff (op1,op2,parent_op)
  (cond ((and (eq op1 op2) (null op1))
          null)                              ;No difference.
        ((eq op1 op2)
         (derivation_diff (derived_op op1) (derived_op op2) op1))
        (t (list parent_op op1 op2))))
derived_correspondences(diff_ap_op,goal1,goal2) ::
 return correspondences((derived_op op1) (derived_op op2)
;;; Returns a list of correspondences: ((op1z . op2z) (op1y . op2y) ...
;;; (op1a . op2a) (op1 . op2)) such that op1 correspondences to op2, op1a is
;;; derived from op1 and correspondences to op2a which was derived from op2 ...
(defun correspondences (op1,op2)
   (if (or (type op1 'primitive) (type op1 'primitive))
       (cons op1 op2)
       (cons (cons op1 op2)
             (correspondences (derived_op op1) (derived_op op2))))))
```

Figure 30. Hierarchical Correspondence Algorithm.

```
(defun find-conflicts (goals1 goals2)
 (loop with other-goals = (loop for g in goals2
                           when (established-goal-p g)
                           collect g)
       for goal1 in goals1
       when (established-goal-p goal1)
       for ctype = (find-most-specific-conflict-type goal1 other-goals)
       collect (make-conflict-record goal1 ctype)))

(defun find-most-specific-conflict-type (goal other-goals)
 (let ((ctypes (find-conflict-types goal other-goals)))
  (or (assoc :POSSIBLE-MEANS ctypes)
      (assoc :OBJECT-ATTS ctypes)
      (assoc :GOAL-ATTS ctypes)
      (assoc :OBJECTS ctypes)
      (assoc :ABSTRACTIONS ctypes)
      (assoc :GOALS ctypes))))
```

Figure 31. Goal Conflict Detection Algorithm.

goals in the second design which have been established.[1] Next, `find-most-specific-conflict-type` compares each goal in the first design against all the goals in the second. It returns a goal and conflict type. The conflict type returned is the most specific way the goals conflict as defined by six conflict types. Figure 32 summarizes these types.

The six conflict types range from no (apparent) conflict to completely dissimilar goals. A possible means conflict denotes that two goals are identical. An object attribute conflict denotes that two goals are identical, except that at least one object varies in its attribute value. A goal attribute conflict denotes that two goals are identical, except that at least one relation attribute varies in value. A goal attribute conflict may also have object attribute conflicts. (The conflict types are subsuming.) An object conflict denotes that two goals have at least one relation object that is different. (In OPIE, these are actually different relations, but with the same names.) An abstraction conflict denotes that two goals have different name, but share a common ancestor in the relation hierarchy. Finally, a goal conflict denotes that two goals have nothing in common.

[1]Goals which have not been established in a design cannot create design conflicts.

Type	Description
Goal	No similarity.
Abstraction	Only similarity is a common ancestor in the relation hierarchy.
Object	Same goal name, but different objects.
Goal Atts.	Differences in the attribute(s) of the goals.
Object Atts.	Differences in object(s) attributes of the goals.
Means	No goal differences, but interfering plans.
Possible Means	No goal differences. Still check for interference.

Figure 32. Table of Goal Conflict Types.

At this point, it may be useful to introduce the specific components that are negotiated. Figure 33 presents a table of components and the ability of the implementation to negotiate with them. For example, Oz detects object attribute conflicts, displays their preferences, and generates their resolutions.

Relations attribute conflicts are detected and displayed, but Oz does not generate resolutions for them. This is simply because they have not been needed for design construction.

Relation conflicts are detected and displayed, but only object conflicts are negotiated. This corresponds to the object conflict type of figure 32. In contrast, one could use the link distance between goals in the relation hierarchy to negotiate about a range of related goals. (In fact, the specialization and generalization methods of section 3.4 do use this technique.) However, there is no means to specify preferences between relations at varied abstraction levels. Instead, only ordering of sibling relations can be used to define relation preferences. Hence, relation negotiations concern only the objects within the relation.

As indicated in figure 33, not all the negotiations available in the representations are implemented in the algorithms. However, this is due to the limitations the project and not of the algorithms. For example, operator conflicts are detected only through their interference. If two goals

Component	Detected	Displayed	Negotiated
operators	some	no	no
operator attributes	no	no	no
relations	yes	yes	objects
relation attributes	yes	yes	no
objects	yes	yes	yes
object attributes	yes	yes	yes

Figure 33. Table Negotiated Components.

are established using different operators, and those operators interfere, then the operator differences will be detected. However, if there is no interference then the operator conflict will not be detected. Operator preferences as a whole have not been implemented in Oz.

2. Interference Determination

After the goal level conflicts have been detected, design level interference is determined. Interference analysis enables one to determine: (1) design level differences from identical goals, and (2) non-interference from "conflicting" goals. It allows the algorithms opportunity to negotiate over the derivation of identical goals and to inform the user that "conflicting" goals can be established in an integrated design without further processing.

Figure 34 presents the interference determination algorithm. It is applied to each goal conflict pair. The function `goal-interference` back propagates each of the conflicting goals through the design level operators which achieve them. An interference structure is created when the design components of the two goals have consuming interference. That is, when at least one scarce relation instance is consumed by each plan.

Interference depends on the state of scarce relation instances. Since it is assumed that both designs have identical initial states, the integrated initial state is identical to either of the given

```
(defun goal-interference(initial-state,goal1,primitives1,goal2,primitives2)
 (loop with preconditions1 = (back-propagate-preconditions goal1,primitives1)
       with preconditions2 = (back-propagate-preconditions goal2,primitives2)
       for relation1 in preconditions1
       when (loop for relation2 in preconditions2
            thereis (and (unify relation1 relation2) ;Using the same relation.
                         (consumed-relation? relation1) ;Consume as opposed to
                         (consumed-relation? relation2) ;just using.
                         (consume-conflict? relation1 relation2 initial-state))
            collect (make-interference-record relation1 relation2))))
```

Figure 34. Interference Determination Algorithm.

designs.[2] Hence, a simple copy operation is used to derive the initial integrated design state. All interference tests are conducted in this initial integrated state. This allows the interference of two goal implementations to be considered in isolation; however, it will not reveal other interference that may occur in the complete design (e.g., other goals may also consume scarce relation instances).

The interference algorithm only considers two-way interference based on the *detect direct interference assumption*:

> **Detect Direct Interference Assumption**
> Direct interference between pairs of goals is: (1) easier to detect, (2) often easier to resolve, and (3) may resolve more dependent exogenous interference than interference between *n* goals, for a given *n*. (If *n* = number of goals, all interference may be resolved, but no exogenous interference will be resolved; that is, no interference is opportunistically removed.)

Two-way interference is easier to detect, because only pairs rather than *n* goals need be applied to the goal interference function. Two-way interference is often easier to resolve, because only the interference of the pair need be considered. Finally, resolving one two-way interference can

[2]If the states were not identical, the initial state would have to be negotiated just as the goals are negotiated. In fact, Oz does apply the goal conflict detection algorithm to the initial design states to check if they are indeed the same.

resolve the of other (exogenous) interference. For example, eliminating one goal which "hogs" *n* items a resource can remove *n* other interferences which simply use one item.

While interference determination only considers two-way interference, a resolution process can also consider *n*-way interference; hence, the direct interference assumption is applied to reduce effort during conflict detection. Conflict resolution may opportunistically resolve conflicts (due to conflict dependencies) as well as further analyze given conflicts, so the interference detection effort should be minimal. I believe two-way interference is an appropriate balance between an "eager" *n*-way and "lazy" *0*-way detection methods.

While the Oz interference algorithm only reveals all two-way interferences, it can easily be modified to consider *n*-way interference. When *n* equals the number of goals, complete interference is considered. In fact, such *n*-way interference is determined when the design process is applied to the integrated perspective.

Figure 35 illustrates the type of records produced by Oz's conflict and interference algorithms. It simply records the goals in conflict, the type of conflict, their derived operators, and any resulting interference. These records are also used by the resolution algorithms; hence, they contain a few more slots. In addition to the original conflicting goals, the current alternative is recorded. Initially, it is set to one of the goals in conflict. After resolution generation, it can be any relation in the relation hierarchy. Finally, the record may also have a plan. This records the a plan that can be used to achieve both goals, if there exists such a plan.

TYPE:	MEANS | OBJECT-ATTS | GOAL-ATTS | OBJECTS | ABSTRACTIONS | GOALS
GOALS:	(G_1, G_2)
DESIGN-OPERATORS:	(Ops_1, Ops_2)
INTERFERENCE:	$(Relations_1, Relations_2)$
ALTERNATIVE:	G_1
PLAN:	$Plan_1$

Figure 35. An Illustrate of a Conflict/Alternative Record.

3. Resolution Generation

Conflict resolution attempts to find acceptable alternatives, given some initial conflicts and perspectives. The Oz resolution approach consists of three major generation methods: compromise, specialization and generation. Additionally, Oz applys replanning to determine if a two-way conflict can be resolved by simply choosing alternative operators.

The replanning method is present next. Then, the interactive framework in which the other three methods are applied, is presented. The actual resolution methods follow.

3.1. Replanning

Before the conflicts are presented to the user, one final analysis of goal conflict is applied. Given two goals with interference, it may be possible to derive a plan which achieves both goals. To check this, each interfering goal pair is passed to the planner. Given two goals and the initial integrated state, the planner attempts to achieve the conjoined goal state. If it can, the goals need not be negotiated. Instead, they can simply be included in the integrated perspective. When the integrated design is derived, both goals may be achieved.

Like interference determination, replanning only considers two-way interaction. This simplification allows the user to quickly determine if is at all possible for the conflicting goals to be simultaneously achieved. If is not, they are deemed inherently conflicting goals (relative to the initial state). On the other hand, if their pair-wise conflict can be resolved through replanning, then the final derivation of a design from the integrated perspective will determine if, in fact, the replanned goals can be achieved in the greater context of all design goals. If not, the goals can be negotiated, a new integrated perspective can be generated, and new integrated design can be derived.

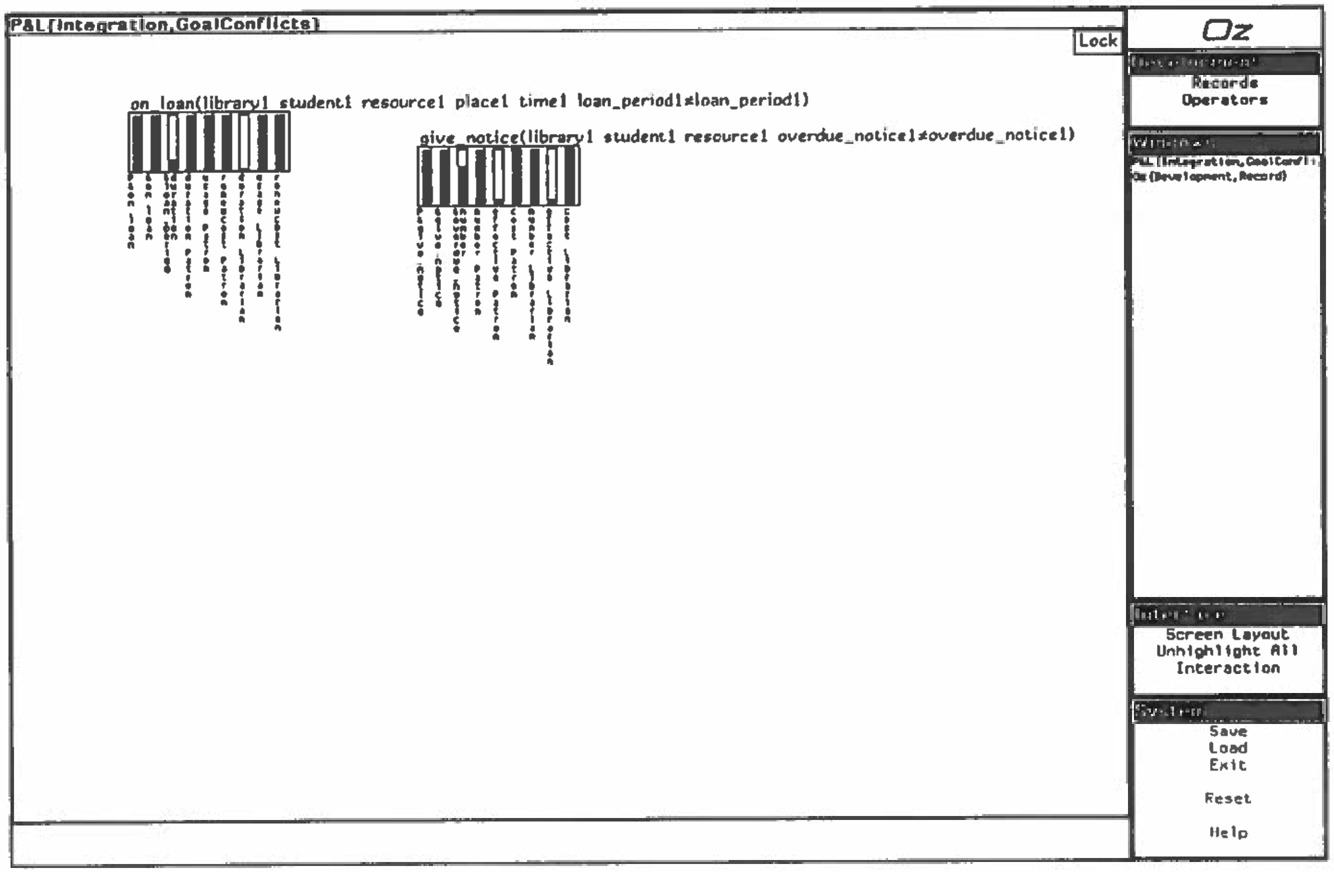
3.2. Interactive Resolution

The interactive resolution framework allows the user to determine which conflict to work on, which resolution method to apply and in which order, and which resolutions are to be accepted. Initially, the user is displayed icons depicting the conflicts. The goal relations are displayed along with any conflicting components. Additionally, all relevant preferences and displayed. The user then applies resolution methods to any of the conflicts to generate alternatives. Those alternatives can, in turn, give rise to still other alternatives via the application of other resolutions methods. Hence, the resolution methods can be interleaved both between different conflicts and within the derivation of a single conflict resolution.

Figure 36 shows an Oz depiction of the initial issues from a conflict in chapter VI. The relations ON_LOAN and GIVE_NOTICE are shown with special attention to the conflicting objects. For example, the ON_LOAN conflict is displayed as:

```
on_loan(library1 student1 resource1 place1 time1 loan_period1≠loan_period1)
```

This indicates that the corresponding objects LOAN_PERIOD1 from the two perspectives are in conflict. (In this case, the two perspectives use the same name; however, it is possible for goals to



```
on_loan(library1 student1 resource1 place1 time1 loan_period1≠loan_period1)
give_notice(library1 student1 resource1 overdue_notice1≠overdue_notice1)
```

Figure 36. Oz Initial Screen Depiction the Goal Conflicts.

match even though they use different object names.)[3]

Preference satisfaction is displayed below the relations. In each case, preferences are displayed in the following left to right order: relation preferences, relation attribute preferences, and object attribute preferences (if they exist). For each of these preference types, first the average of the perspectives is displayed, and then preferences from each perspective is displayed (if they exist).

Reading from right to left, consider the preferences associated with ON_LOAN. The three rightmost are the librarian's object attribute preferences for LOAN_PERIOD1. The loan period attribute preferences are: resource cost, usage, and duration. The next three preferences toward the left are the patron's object attribute preferences for LOAN_PERIOD1. Again, the loan period attribute preferences are: resource cost, usage, and duration. The next preference indicates the average satisfaction of all the LOAN_PERIOD attribute preferences, i.e., resource cost, usage, and duration from both perspectives. This averaged preference is named &LOAN_PERIOD DURATION. Loan period is the object and duration is the attribute whose value differed in the two designs. The next preference to the left is the averaged ON_LOAN attribute preference. Since there are no attribute preferences associated with the ON_LOAN relation, no individual preferences are displayed. However, notice that its satisfaction is at 100 percent. Whenever there is no preferred ordering associated with a range, its satisfaction is displayed at 100 percent, thereby allowing the arbitrator to effectively ignore its contribution to an alternative's overall achievement. Finally, the last preference to the left is the averaged ON_LOAN relation preference. Since there are no relation preferences associated with ON_LOAN, no individual preferences are displayed. The

[3]In the case of conflicting relations or relation attributes, the relations would be displayed as conflicting. For example, GIVE_NOTICE≠GIVE_NOTICE. (Refer to section 1.3.)

GIVE_NOTICE preferences are displayed in similar fashion.

Each displayed relation represents an alternative. Hence, an initial relation represents two concepts. It represents the initial conflict, where the display notes the conflicting components with the ≠ symbol. And, it represents an alternative where the conflict is resolved. In the case of the initial relation, this conflict is resolved by choosing the values from the first perspective given to the integration; the value from the patron's perspective. Hence, the first relations displays the conflicts and the alternatives where the first perspective's values are chosen. Subsequent alternatives will be generated and displayed using the same format. However, they will only represent resolutions; hence, no conflict will be displayed.

For each relation (alternative) displayed, preference satisfaction is displayed in the form of a bar graph. The more satisfied a preference, the higher the bar graph. Hence, one can see from the initial relations displayed the preferences from the two perspectives on the proposed resolution of simply one of the conflicting values. Thus, one set of preferences are higher than the other for each relation.

3.2.1. Resolution Display

The resolution display may be simple enough that observation may be sufficient to understand it. However, it may help to expand on the interaction of preferences during search. Here, is a simple example.

Below are two preferences from two perspectives (per-1 and per-2).

```
(in-perspective 'per-1)
(def_mod_relation I.1
  :order-by-value '((A 20) (B 30) (C 100)))
```

```
(in-perspective 'per-2)
(def_mod_relation I.2
  :order-by-value '((A 50) (B 5) (C 100)))
```

```
Alternative a¹ = (A)
Alternative a² = (B)
Alternative a³ = (C)
```

Also, the initial alternative, a^1 is A. Each alternative is evaluated by the preferences from each perspective: $(I.1(a^1), I.2(a^1)) = (20,50)$. Figure 37 illustrates these preferences. All alternatives are shown in both parts, with the alternative being considered in the foreground. Part (a) of figure 37 displays a^1 = (A). Part (b) displays a^2 = (B).

Parts (a) and (b) of figure 37 are snapshots of an interactive search. Initially, alternative a^1 and its preferences are displayed. Next, the user attempts to increase the satisfaction of preference I.2 from 5% to 50% by moving it to the foreground. Unfortunately, that alternative decreases the satisfaction of I.1. If the user continues on to a^3 = (C), the ideal is reached; i.e., the best value from each perspective.

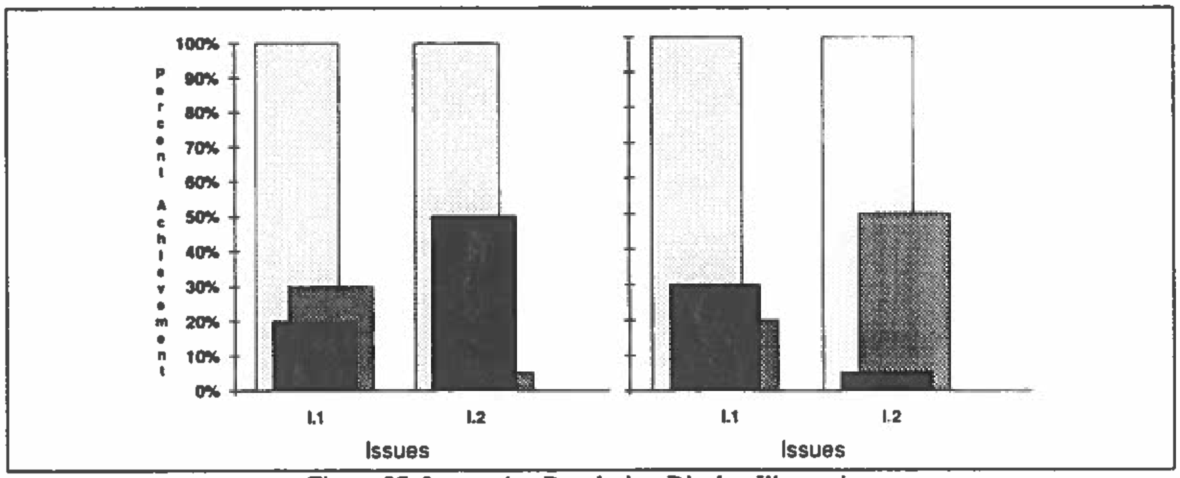


Figure 37. Interactive Resolution Display Illustration.

This search for the ideal was introduced in Chapter I, section 1.7. Unlike Zeleny's ideal, or the above example, Oz resolution search does not have a mechanism to track the user's focus on alternatives. Instead, all alternatives for a given resolution method are displayed at once. User's can then continue search by choosing an alternative to be further negotiated, or end search by choosing the final alternative.

3.2.2. Preference Modification

Preferences can be modified during the resolution search process. To do so, the user selects an alternative, modifies the preferences, and the display of that alternative is redrawn based on the new preferences.[4] However, preferences are global. Hence, changing preferences during resolution may result in inconsistent decisions: the initial design process imposing one set of preferences while resolution process derives another. Such inconsistency will be overcome during the resolution implementation process. During that process, all the current preferences, as determined during the negotiations, will be used to derive an integrated design.

3.3. Resolutions Search

After an initial nondominated space is created, the user participates in resolution search. Figure 38 presents `interactive_search`; it's simply a loop enabling five mousable commands. The user is assisted in search via graphic depiction of the search space and the current alternative. This guides the user's choice of accepting the current alternative, altering preferences, or generating resolutions. After any modification, a new display is rendered.

[4]This modification method is very primitive.

```
interactive_search()
 loop
   for mouse_click = get_mouse()
     case: mouse_click = compromise
           compromise(current_alternative)
     case: mouse_click = specialize
           level = get_level()
           specialize(current_alternative,level)
     case: mouse_click = generalize
           level = get_level()
           generalize(current_alternative,level)
     case: mouse_click = select_alternative
           current_alternative = select_alternative
     case: mouse_click = preference_edit
           preference_edit(current_alternative)
     case: mouse_click = choose_alternative
           mark_resolved(current_alternative)
 until mouse_click = accept
return marked_alternatives()
```

Figure 38. Interactive Resolution Search Algorithm.

3.3.1. Choosing a Resolution

While Oz does provide a interactive search framework, it does not provide a strategy. Zeleny's *displaced ideal* does suggest one strategy. It uses the current alternative to define a search space and a goal to obtain. The goal is the (infeasible) composite of each preference's maximum achievable value within the feasible alternatives, i.e., the best known value for every preference without any of the negative interactions.

> As all alternatives are compared with the ideal, those farthest away are removed from further consideration. There are many important consequences of such partial decisions. First, whenever an alternative is removed from consideration there could be a shift in a maximum attainable score to the next lower feasible level. Thus, the ideal alternative can be displaced *closer* to the feasible set. Similarly, addition of a new alternative could displace the ideal *farther away* by raising the attainable levels of attributes. Such displacements induce changes in evaluations, attribute importance, and ultimately in the preference ordering of the remaining alternatives. — p. 143[92].

Figure 39 illustrates the displaced ideal model.[5] Alternatives x^*, x^{**}, and x^{***} are consecutive instances of the ideal alternative. Initially, x^* is the ideal defined from the extreme positions along each preference. However, as x^1 and x^2 are removed from consideration, the ideal changes to be x^{**} and x^{***}.

Zeleny's interactive search procedure is comprised of three basic tasks (cf[92].). Based on the displaced ideal theory, it makes use of an interactive display to guide search.

(1) Seek Ideal

Initially, the worst alternative is displayed—the anti-ideal. Next, the user seeks the ideal by considering increased satisfaction of important preferences. As the ideal is sought, dependencies between preferences can be observed. Eventually preferred alternatives will become familiar.

(2) Identify Cycle

The user considers various alternatives, several will reoccur; a cycle is observed. Members of this cycle depict a narrow set of alternatives, one of which may be chosen.

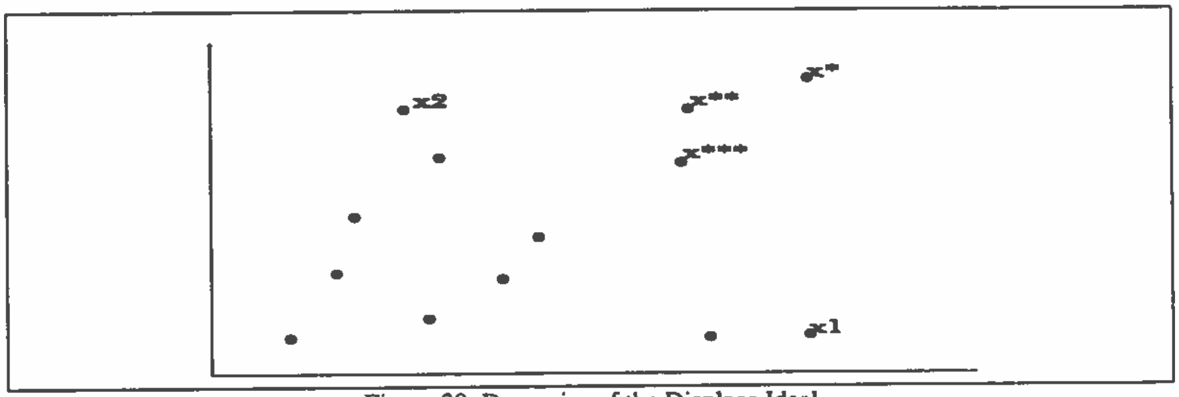


Figure 39. Dynamics of the Displace Ideal.

[5]Derived from figure 3-1 in[92].

(3) Apply Tests

Test conducted exterior to the search procedure can be used to aid choice. Additionally, preferences can be modified.

(4) Apply Heuristics

If the current alternatives reveal no acceptable alternative, new alternatives must be sought. The generalization and specialization methods assist this process. Next, the user begins again at step one, until a satisfactory resolution is obtained.

While these four steps form the basis of displaced ideal strategy, one can use several variations; however, any strategy may be applied—strategies are not enforced during interactive search. A variation of step 3 allows users to skip tests and immediately apply heuristics (step 4). Another variation allows users to start at the closest alternative to the one displayed when moving from steps 3 to 4, rather than starting at the anti-ideal again.

3.4. Generation Methods

Given a conflict, alternative generation finds possible resolutions. Using preferences and goal hierarchy to characterize conflicts, **Oz** is able to use three methods of conflict generation. *Compromise generation* maximizes multiple criteria given linear constraints. *Specialization generation* presents more specialized resolutions from the relation over which the conflicting values can be distributed; thereby, removing the conflict. Finally, *Generalization generation* presents more abstract relations from the relation hierarchy to: (1) remove the conflict through abstraction, or (2) provide alternative compensation for the "loser" of a negotiation. Next, the three generation algorithms are presented.

3.4.1. Compromise Algorithm

The compromise algorithm is a linear multi-function maximization solver. One can define l linear functions or criteria to be maximized as $f_l(x) = \Sigma_{j=1}^{n} c_{ij}x_j$:

$$
\begin{aligned}
&f_1(x) = c_{11}x_1 + c_{12}x_2 + \ldots + c_{1n}x_n \\
&\ldots \\
&f_l(x) = c_{l1}x_1 + c_{l2}x_2 + \ldots + c_{ln}x_n
\end{aligned}
$$

The linear criteria can be subjected to m linear constraints $gr(x) = \Sigma_{j=1}^{n} a_{rj}x_j = b_r$, $r = 1 .. m$, $x_j \geq 0$:

$$
\begin{aligned}
&g_1(x) = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
&\ldots \\
&g_m(x) = a_{l1}x_1 + a_{l2}x_2 + \ldots + a_{ln}x_n = b_m
\end{aligned}
$$

Constrained objectives defined in this way are amenable to linear programming methods. Specifically, the linear simplex method can expanded to account for multiple objectives. It will terminate after finding all nondominated extreme point solutions, i.e., after maximizing (or minimizing) each function, f_i.

Converting the represented preferences to the simplex notation is simple. The preference objectives become the functions to be maximized or minimized.[6] Preference orders becomes the ranges over which the functions are to maximized. If each component is not given a specific value (using `:order-value`), then values are assigned in even increments between 0 and 100. For any preference, all perspectives must use the same range of values. For example, if one perspective

[6]Goal seeking is also implemented. Then, the objective is to minimize the distance from a specific value.

represents loan period duration from 0 to 365, all perspective must use the same range.[7] Hence, ranges, and constraints in general, are not subject to negotiation. This limitation is due to the use of the simplex method. Discrete values is another shortcoming of the method. The simplex method only searches on continuous ranges. Given a discrete alternative, one must truncate it to the nearest discrete value.

Next, the compromise algorithm is illustrated with a simple library example. A more complete example is available in appendix B.

The object LOAN_PERIOD has three attributes: duration, cost, and risk. Both cost and risk each rise linearly with duration. Also, the combined value of cost and risk must not be above 100; however, their they are combined according to the constraint: $0.5 * \text{COST} + 0.8 * \text{RISK} \leq 100$. Given these constraints, the librarian wants to maximize duration, while minimizing cost and risk. In contrast, the patron simply wants to maximize duration. The following two perspective represent this problem.

```
(in-perspective 'Librarian)

(def_mod_object "loan_period"
 :attributes '(("Duration" :min 0 :max 365 :obj max)
                ("Cost" min 0 :max 100 :obj min)
                ("Risk" :min 0 :max 100 :obj min))
 :att_constraints '(
                 (0 0.5 0.8 <= 100)     ; Limit risk & cost
                 (1 -1 0 = 0)       ; Cost = Duration
                 (1 0 -1 = 0)))    ; Risk = Duration
```

[7]Varied ranges would not cause an error for the system. However, it would use the smallest range to define the search space.

```
(in-perspective 'Patron)

(def_mod_object "loan_period"
 :attributes '(("Duration" :min 0 :max 365 :obj max)
                ("Cost" min 0 :max 100 :obj nil)
                ("Risk" :min 0 :max 100 :obj nil))
 :att_constraints '(
                (0 0.5 0.8 <= 100)      ; Limit risk & cost
                (1 -1 0 = 0)     ; Cost = Duration
                (1 0 -1 = 0)))   ; Risk = Duration
```

This problem can be represented in simplex notation as:

$$
\begin{array}{llcrcrcr}
\text{Max} & \text{DURATION}_{\text{PATRON}} & & & & & & \\
\text{Max} & \text{DURATION}_{\text{LIBRARIAN}} & & & & & & \\
\text{Min} & \text{COST}_{\text{LIBRARIAN}} & & & & & & \\
\text{Min} & \text{RISK}_{\text{LIBRARIAN}} & & & & & & \\
 & & + & 0.5\ \text{COST} & + & 0.8\ \text{RISK} & \le & 100 \\
 & \text{DURATION} & + & & + & & \le & 100 \\
 & & + & \text{COST} & + & & \le & 100 \\
 & & + & & + & \text{RISK} & \le & 100 \\
 & & & & & \text{DURATION} & \ge & 0 \\
 & & & & & \text{COST} & \ge & 0 \\
 & & & & & \text{RISK} & \ge & 0
\end{array}
$$

Figure 40 illustrates the search space for loan period durations. Cost and risk depend linearly on duration. This results in the line described by $x = y = z$; however, it is constrained to a height of 76.92%. Compromise resolutions are real values triples running from 0 to 76.92: (0,0,0), (1,1,1), ... (76.92,76.92,76.92); the arbitrator must balance the conflicting objectives to determine which resolution will be chosen. The multiple criteria simplex method (MCSM) only generates the extreme points of search spaces (e.g., (0,0,0) and (76.92,76.92,76.92); however, all compromise points can be obtained from the extreme points[92].

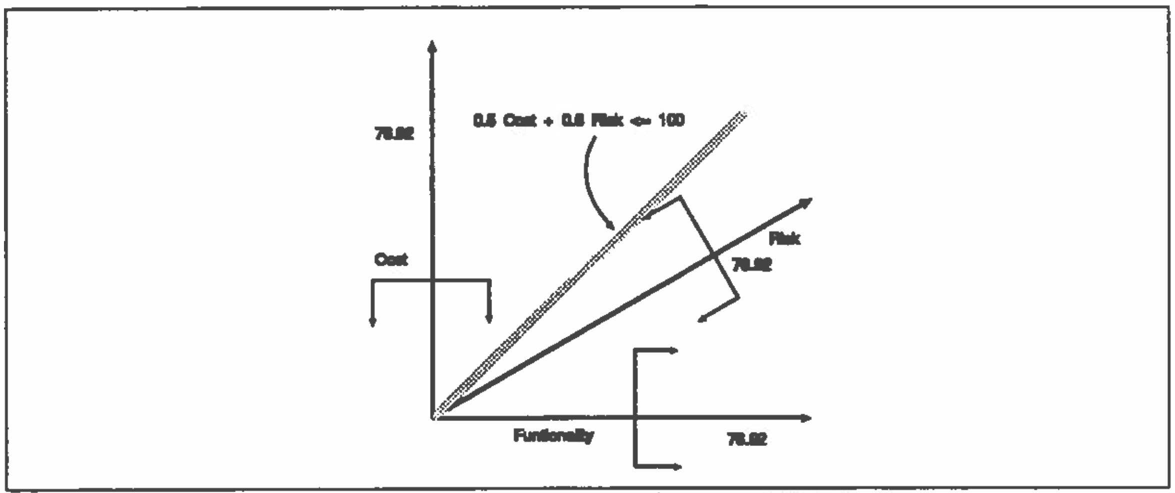


Figure 40. The Nondominated Search Space of LOAN_PERIOD with Risk and Cost.

3.4.2. Specialization and Generalization Algorithms

The specialization and generalization algorithms find similar relations to one under consideration. They are complimentary algorithms which search down and up both the relation and operators hierarchies. New specialized relations can used to create a dissolution of a conflict: each conflicting value can be distributed over some specializations. New generalized relations can be used to create a dissolution of a conflict, or create a compensation for poorly satisfied preferences. A dissolution is created by replacing the conflict over values with a single abstraction. A compensation is created by adding a new goal to the integrated design which achieves some satisfaction for preferences which loose out in the original conflict negotiation.

The specialization and generalization algorithms are directly complimentary. Their major difference is the direction of their search. Specialization search down the hierarchies, while generalization searches up.

The relation and operator hierarchies serve as the basis for determining similar relations. Both hierarchies are based on the abstractions of components. Components higher in the hierarchies are more abstract than those below. For the relations hierarchy, abstraction is based on object. For the operator hierarchy, abstraction is based on operator add and delete lists, i.e., functionality.

Given a relation which is the focus of resolution, one can find closely related relations by moving a few links up and down the relation hierarchy. Similarly, one can find operators which produce a given relation and then move up and down the operator hierarchy and find similar relations on the add lists of the operators. These operator relations are not so directly related to the given relation as those found in the relation hierarchy. That is, they may not be directly related through object abstraction. Instead, they are related through functionality. Operators which produce similar relations are near each other in the hierarchy. Hence, relations found by searching indirectly through the operator hierarchy may be related functionally, but not through object abstraction.

The algorithms use these two hierarchies as the basis for generating resolutions. The given relation is the entry into the relation hierarchy. The operators which produce the given relation are the entry points into the operator hierarchy. (For specialization, the root of this tree is the entry point; for generalization, the leaves of the tree are the entry points.) Once, these entry points are established, the two methods diverge. Specialization moves down the hierarchies, while generalization moves up. Both algorithms are parameterized by the number of links they traverse.

Figure 41 presents both algorithms as one where the direction of movement is also taken as an parameter. The function `sg-resolutions` applies the two functions `relation-hierarchy-relations` and `operator-hierarchy-relations` and then packages those relations into alternative resolutions. The function `relation-hierarchy-relations` simply returns related relations. However, the function `operator-`

```
(defun sg-resolutions (relation1 n direction)
  (resolution-groups relation1 n direction
                  #'relation-hierarchy-relations
                  #'operator-hierarchy-relations))

(defun relation-hierarchy-relations (relation1 n direction)
 (nth-ancestor relation1 n direction))

(defun operator-hierarchy-relations (relation1 n direction)
 (mapcan #'(lambda (operator) (nth-ancestor operator n direction))
       (all-added-by-ops relation1 direction)))
```

Figure 41. The Specialization/Generalization Algorithm.

`hierarchy-relations` return sets of relations. These relation sets are all those relations on the a related operator's add list minus all those relations added by the operator which added the given relation. Hence, each set indicates the new relations which will be produced if the given resolution is accepted. That is, if the new relations are accepted, then the new operator will assert new relations listed. For example, given the ON_LOAN relation, the OWN(AGENT1 RESOURCE1) relation is suggested through the BUY_RESOURCE operator. However, OWN(AGENT1 MONEY1) is also part of that relation set, since OWN(AGENT1 MONEY1) is not a member of BORROW_RESOURCE's add list (the operator which asserted the given ON_LOAN relation). Figure 42 shows an Oz depiction of this generalization.

4. Resolution Implementation

Resolution implementation derives an integrated perspective and design from given perspectives and resolutions. The basic approach is to copy one of the perspectives and transform it using the resolved goals. The resulting perspective is then given to the planner, which then derives the integrated design.

In the simplest case, the original goals from one perspective can be simply transformed to produce the integrated goals. The simple formula for this is:

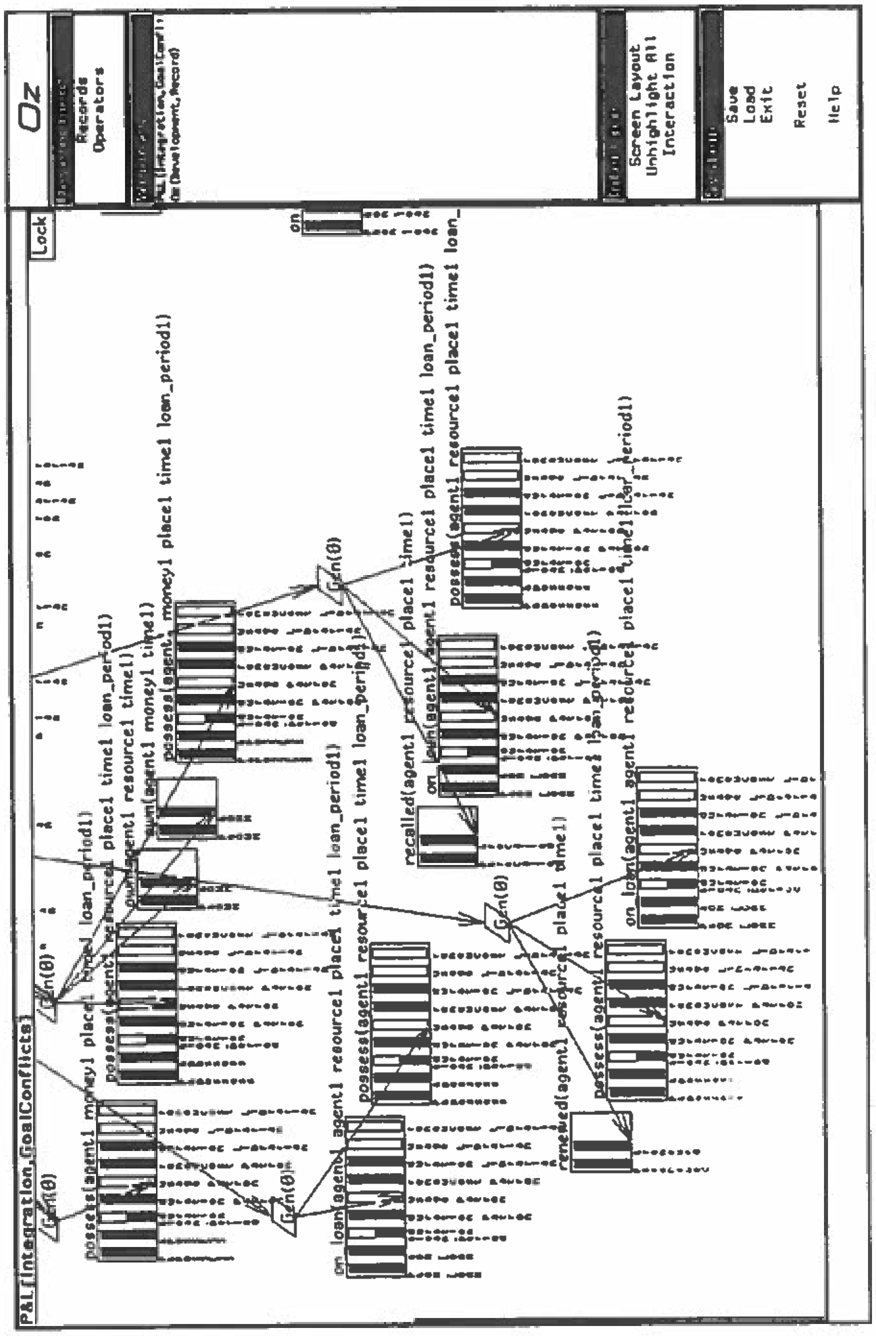

Oz
Records
Operators
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help
Lock
P&L [Integration,GoalConflicts]
Gen(0)
possess(agent1 money1 place1 time1 loan_period1)
possess(agent1 resource1 place1 time1 loan_period1)
sum(agent1 resource1 time1)
sum(agent1 money1 time1)
possess(agent1 money1 place1 time1 loan_period1)
on loan(agent1 agent1 resource1 place1 time1 loan_period1)
possess(agent1 resource1 place1 time1 loan_period1)
recalled(agent1 resource1 place1 time1)
on loan(agent1 agent1 resource1 place1 time1 loan_period1)
possess(agent1 resource1 place1 time1 loan_period1)
renewed(agent1 resource1 place1 time1)
possess(agent1 resource1 place1 time1 loan_period1)
on loan(agent1 agent1 resource1 place1 time1 loan_period1)

Figure 42. Oz Screen Depiction the of Generalizations (2).

initial goals - conflicting goals + resolved goals

This transformation replaces the conflicting goals with the resolved goals. All nonconflicting goals are also carried into the integrated perspective.

To make matter more difficult, each goal contains specific referents. For example, the following goal refers to specific objects.

```
on_loan(library1 student1 resource1 place1 time1 loan_period1)
```

Hence, when new relations are introduced during resolution, they must use the proper referents. For compromise and relation specialization, referents can easily be obtained from the given relation. For example, if the above relation is specialized to:

```
on_loan(library1 graduate1 resource1 place1 time1 loan_period1)
```

the student1 referent can be replaced with the graduate1 referent. However, any other relation instances which refers to this referent must also transformed. This means that the initial state must have its referents updated. For example, the following transformation must take place:

```
loan_period(student1 resource1 place1 time1 loan_period1)
☞
loan_period(graduate1 resource1 place1 time1 loan_period1)
```

When generalized relations are introduced during resolution generation, the transformation process must use heuristics to determine the proper referent. Hence, sometimes the transformed perspective must be manually modified before being passed to the planner. For example, if two OWN relations are introduced as a generalization of ON_LOAN to indicate resource selling (i.e., ownership of the buyer's money and ownership of the seller's resource), then the transformation process must guess to determine which referent will provide the money (student1) and which referent will provide the resource (library1). However, if the relation instances matching the preconditions of the given relation's operator are unified with the preconditions of the suggested relation's operator, one may obtain a good substitution of referents. Oz does not use this method. Instead, it uses simpler heuristics and the aid of a human.

Once the referent substitution are propagated through the copied perspective, it can be given to the planner. The planner then attempts to achieve the integrated goals. When each goal interference is confined to corresponding goal conflicts (e.g., two-way interference) and the integrated goals are specializations of the original goals, the planner is guaranteed to derived an integrated design. However, if two corresponding goals not only interfere with each other, but with other conflicting goals (e.g., they all consume a single relation instance), then the planner may not be able to achieve an integrated design. Additionally, if the goals were transformed through generalization, then the planner may not be able to achieve an integrated design. (There may be no available operator, or the goal may interfere with other goals.)

5. Evaluation

This section evaluates the integration methods. Evaluations consists of (1) assumptions, (2) computational analysis, (3) and related research. First, conflict detection is evaluated. The three resolution generation methods follow.

5.1. Conflict Detection

Recall conflict detection characterizes design differences. The following evaluation considers the complexity of the method. Comparison to others research illustrates the lack of research in this area.

5.1.1. Analysis

Any variation in goals or their derivation is a goal conflict. Goal conflicts are characterized by correspondences, component conflicts, and interference; only two-way inter-design goal interference is considered.

The issue formation algorithm relies on two key assumptions: (1) *derived components correspond* and (2) *only detect direct (two-way) interference.* Within the confines of **Oz**, assumption 1 cannot be incorrect. (If it is, it implies that the planner is malfunctioning.) When assumption 2 is invalid, aggregated multi-goal *interference* goes undetected; however, its *conflict* is known. The resolution process is given goals as conflicting, but has knowledge only of two-way *interference*. *n*-way interference detection is deferred until integration time.

In some cases, deferring *n*-way interference detection will be more efficient. However, if every goal interferes with every other goal, it will be more efficient to aggregate all goals before designing. If every goal is independent (planning's linearity assumption), then it is more efficient to construct independent designs and combine them. No doubt, many problems are somewhere between. Multiple Perspective Design (MPD) methodology calls for independent designs, mainly on grounds of individual preferences accuracy and parallel efficiency; however, if every goal interferes with every other, it will be less efficient.

Given the assumptions, conflict detection turns on the number of conflicts. Figure 30 presented the issue formation algorithm. Its main routine consists to two loops which compare the goals of two perspectives. This results in an overall complexity of O(m*n); where m and n are the number of goals in the two perspectives. After the goals group into conflict pairs, the `goal-interference` function is applied. It must back propagate preconditions of the goals in conflict. One can assume this is some constant p linearly associated with the maximum subplan length. However, each interfering pair is also given as input to the planner to determine if the conflict can be solved by replanning. Hence, this exponential process adds considerable complexity to the interference determination process. Since **Oz** does not have control over the number of conflicts, it can best reduce computation through by using an efficient replanning method.

5.1.2. Related Research

This section presents software engineering and artificial intelligence methods for issue formation. In both case, but especially SE, conflict detection and resolution and hardly separable. These approaches do not have conceptual distance between interference and its characterization; hence, they only implicitly address issue formation.

5.1.2.1. Software Engineering

The simplest way to conduct interaction analysis is to make it unnecessary; this is common in programming methodology. Prior to building a large program, interfaces between components are specified. After the components are designed, they can be "merged" without ill affects[19]. Unfortunately, such a method does not address how the initial interface specifications are derived; presumably, it entails trial and error involving conflict resolution.

Other methods are based syntactic differences and their implied behaviors. The simplest method is the Unix *diff* program;[8] it outputs differences based on string comparisons. This method can be improved by a semantic program model or knowledge of difference relationships.

[8]Unix is a Trademark of Bell Laboratories.

Such knowledge is beyond typical language-oriented module combining techniques which prevent conflict or specify resolution using syntax-based rules[11, 33, 34, 43, 75]. For example, Horwitz et. al. use data and control relations to guide their merging of modifications of a simple base program[38]. Similarly, Berzins uses a general model of program semantics to merge applicative programs[7]. However, both methods assume *behavior aggregation* (i.e., the merge must include all behaviors specified) and neither address conflicts once found. These methods are not able to resolve conflicts because the intentions of the designers are not available. For example, consider the merge of program A with behaviors a_1 and a_2 and program B with behavior b; the designer may wish to merge the programs while excluding a_2. Both methods could handle this merge if such merge relations could be expressed. However, more explicit conflicts such as Merge($b, \neg b$) require an understanding of the goals from which the behaviors were derived and understanding of the intent of the merge.

The above software methods do not address issue formation because (at most) they only consider the program semantics; these do not consider programmer intent (i.e., the program specification). Issues formation requires conceptual distance between conflicts and issues. AI planning is a computational paradigm which does allow some separation between issues (goals) and programs (plans).

5.1.2.2. Planning

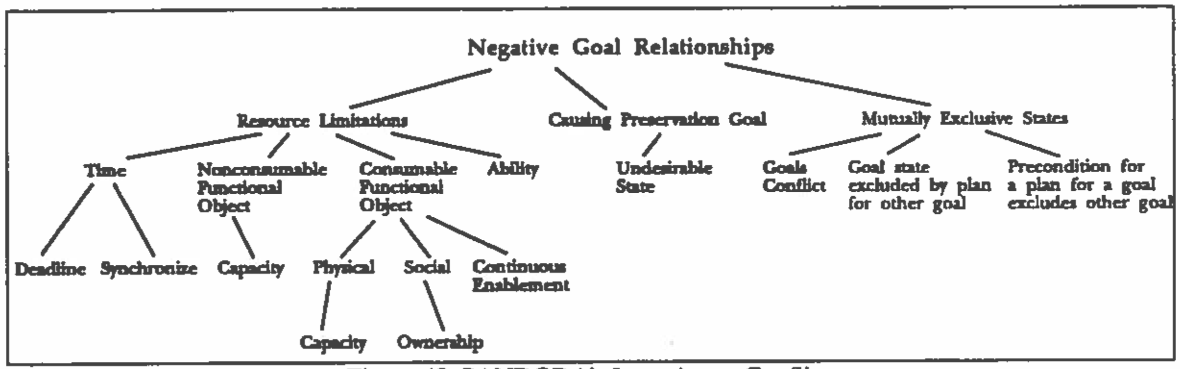
AI planners link behaviors to goals, if only implicitly. Most planners do so through simple goal/operator representations; however, other AI systems represent goal interactions as explicit knowledge.

For AI planning systems, interaction analysis is simplified through representational assumptions. Goals are simple state descriptions. Operators can modify states when their *preconditions* (state descriptions) are met; when applied, predicates are *added* and *deleted* to/from a state description. Goal interactions are completely described by the available operators. The closed world assumption makes this possible: all effects of an operator application are described in the effects list of an operator[64]. Hence, goal interactions are implicitly specified in the effects of operator combinations that can achieve them. If all possible operators (and their effects) could be completely specified such that their combined effects could be predicted, then one would have a complete model of goal interactions. Instead, planners rely on approximations and prepare for exceptional executions[1, 60]. In most cases, they resolve interference between operators during plan formation[14].

Some planners use relationships between operators to aid modeling. Alterman uses category, partomic, causal, and role knowledge to describe plans[1]. Anderson's OPIE derives such hierarchies from operator descriptions[2]. Given a plan, operators and objects are abstracted and added to operator and object hierarchies, respectively. Again, goals are identified with preconditions on operators. However, abstract goals can match with abstract operators; moreover, the operator hierarchy implicitly represents the interactions between abstract goals in the abstracted effects of operators. Such abstract goal interactions can be made explicit[37]. Hammond's CHEF infers and generalizes negative interactions which lead to plan failure; they form demons which anticipate plan failures[36].

Despite such planning successes, Wilensky's criticism still holds.

> ...most AI planning systems do little reasoning about goals and emphasize instead the production of plans.— p. 13[87].


Negative Goal Relationships
Resource Limitations
Causing Preservation Goal
Mutually Exclusive States
Time
Nonconsumable Functional Object
Consumable Functional Object
Ability
Undesirable State
Goals Conflict
Goal state excluded by plan for other goal
Precondition for a plan for a goal excludes other goal
Deadline
Synchronize
Capacity
Physical
Social
Continuous Enablement
Capacity
Ownership
Figure 43. PANDORA's Intra-Agent Conflicts

To see this, simply compare the languages of Wilensky's PANDORA (figures 43 and 44) and Hammond's CHEF (figures 45 and 46). CHEF's understanding of conflict, like most planners, is

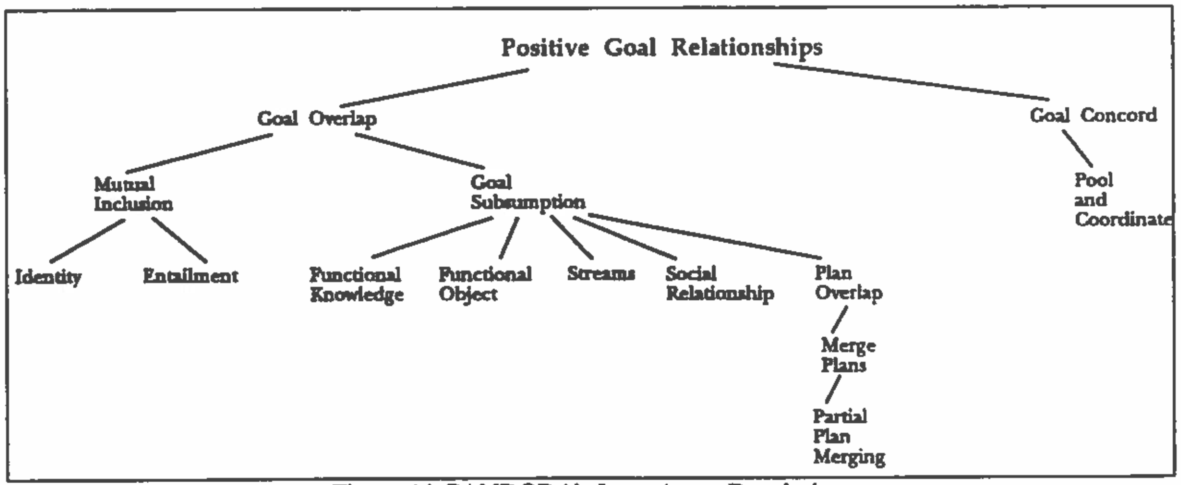


Figure 44. PANDORA's Intra-Agent Resolutions

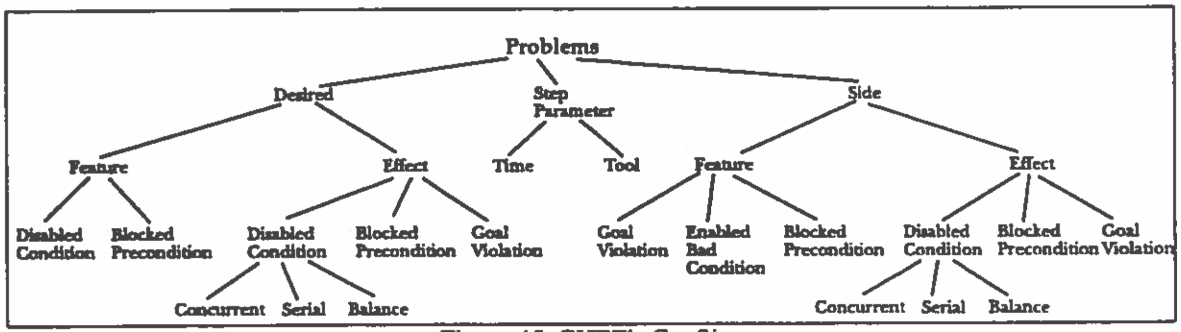


Figure 45. CHEF's Conflicts

based on plan modification operators: (1) alternative operators, (2) reordering, (3) reestablishment, and (4) separation (i.e., alternative unification)[14]. Issue formation is tied directly to the current plan; plan *interference* is the issue. Hence, conflicts and resolutions are described in terms of enablement of operators. The larger context of goal importance, goal relaxation, and the intent of the planner is lost. In contrast, PANDORA's conflict descriptions contain causes behind operator enablement, e.g., resources, goal entailment; PANDORA's resolution descriptions not only consider plan alteration, but environment and goal alteration. Hence, conflicts and resolutions consider more than just the plan, but the attributes of the environment and the overall attitude of the planner.

Most planning systems are concerned with efficiently constructing a plan to achieve a goal; search combined with delayed commitment to operator sequence, object selection, and operator selection is the basic method[2]. Unfortunately, most real world planning calls for the achievement of multiple goals; moreover, such goals fulfill higher goals. Wilensky's theory of metaplanning attempted to address such common-sense planning[87]. However, while PANDORA does have notions of goal importance, partial fulfillment, and abandonment, its reasoning about goal trade-offs is unspecified. In fact, only recently have its basic ideas been brought to fruition (e.g., reasoning about conflict between episodic goals and long term goals)[59].

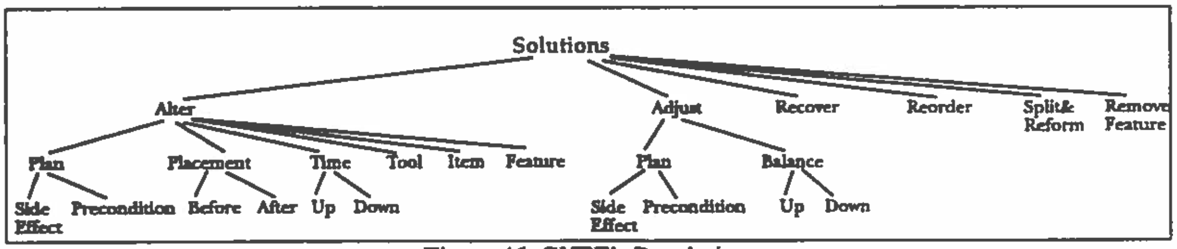


Figure 46. CHEF's Resolutions

Others are linking agent intent to planning and plan modification. In her modeling of a labor mediator, Sycara combines models of goal interaction and agent belief[83]. Figure 47 illustrates PURSUADER's model of a company's goals and their relationships. Goals lower in the hierarchy are supporting or detracting factors to goals higher up. For example, PROFITS are affected by PRODUCTION COSTS and SALES. The company wishes to increase profits (indicated with a "+"); the company believes profits can be increased by decreasing PRODUCTION COSTS and increasing SALES. The graph models agent goals and how the agent believes the goals support or detract from each other.

Graphs like that of figure 47 have been used by organizational[76], decision[90], and negotiation researchers[20]. They typically contain causal relations, markings of importance, and sometimes probabilities. The graphs are drawn from an agent's beliefs, hence the causality may be faulty. Such goal knowledge can be used to prevent the pursuit of incompatible goals, explain why multiple goals cannot be met, or persuade an agent to accept relaxed goals.

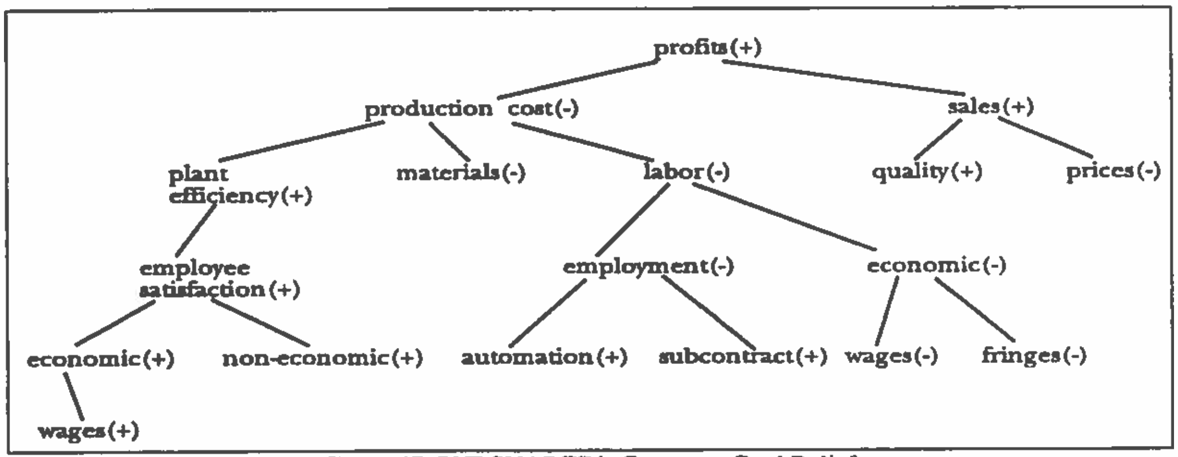


Figure 47. PURSUADER's Company Goal Beliefs

Analogy researchers also consider goal level knowledge. In particular, they are concerned with linking lower level language interactions to higher level goal relationships; this facilitates the determination of a good analogy[12, 31, 35, 44]. Analogical reasoners must resolve conflicts between domains. They decide which structures should be transferred between domains. Unlike planners, most analogical reasoners have some measure of fit; analogies can be made despite constraint violations. Better analogies are achieved by reducing constraint violations. Similar to planning, operators pick alternative structures, alternative unifications, and reorder structures. Analogical reasoners do not use goal interaction knowledge, such as in figure 47, to guide their analogies. Typically, syntactic fit guides this process[31]; however, functional knowledge is also used[44].

In sum, research has focused on plan formation while neglecting issue formation. This focus stems from a single agent view with *few* fixed (presumably achievable) goals. Such myopia naturally focuses on plan patching. When one's planning view is expanded, low level interference must characterized and associated with a subset of the many goals. Similarly, plan patch failure leads to expanded consideration of resolution. Resolution must include goal relaxation and dropping; moreover, preferences must be available to make such decisions.

5.2. Resolution Evaluation

This section presents a broad view of the complexity of the three generation methods. For resolution, computation depends on the issues, perspectives, heuristics, and hierarchy depths. However, the central most determiners are the number of conflicts and the size of the domain hierarchies.

5.2.1. Compromise: Multi-Criteria Simplex

Unlike the other search methods, the compromise algorithm is simply the adaptation of another's research. Hence, this section simply summarizes the analysis of the multi-criteria simplex method. See the references for future details.

Consider a linear programming problem with 10 variables each with a range of 10 integers (e.g., 0 .. 9); the solution space would be 10^{10}. A simplex tableau with n nonnegative unknowns and m dependent variables (constraints) has n!/m!(n-m)! basic solutions; however, most of these "solutions" do not exist[78].

While the simplex method is exponential, computation time is nearly proportional to the number of iterations. The number of iterations depends on the topology of the extreme points. If one only considers extreme points of the feasible set, then the maximum number of extreme point solutions is $X_{ex} = m(n - m - 1) + 2$[46]. Once an extreme point is identified, others can be easily identified by transforming the tableau; this means $m(n - m - 1) + 1$ is the worst case number of simplex iterations.

The multi-criteria simplex method has essentially the same complexity as the single criteria simplex method. However, it limits itself to considering only nondominated extreme points; the nondominated set of extreme points is usually less than the feasible extreme points, $N_{ex} < X_{ex}$. To test for dominance, it must execute phase II of a two phase simplex method on the criteria row; see Zeleny for the exceptions and a way around this[93]. Also, the MCSM has more overhead associated with finding and executing pivoting procedures. Moreover, it must visit all nondominated extreme points, whereas the simplex attempts to maximize one objective by finding one optimal point. See Zeleny or Yu for proofs of completeness and termination[91, 93].

While the MCSM is exponential, Zeleny does point to a polynomial algorithm (Appendix B[92],). In practice, the MCSM is adequate for most problems. The following table gives the possible solution space, nondominated extreme point solutions, and CPU non-system run times for several problems appearing in[92]; and[93] the (nonoptimized) algorithm was on a Sun4 in Allegro common lisp.

Year	Page	(n, m)	Space	Nex	Time (sec)
1982	233	(5, 3)	10	2	0.034
1982	244	(7, 4)	35	6	0.366
1974	116	(16, 8)	1,2870	3	0.183
1974	117	(16, 8)	1,2870	281	11.866

Despite the computation adequacy of the MCSM it lacks several important features. In general, a good constraint system which has the following features:

Qualitative

In addition to discontinuous ranges, discrete ranges should be expressible[8].

Multiple (Alternative) Constraints

Alternative sets of constraints should be expressed and efficiently searched. This means expanding search to constraint spaces rather than a single constraint space. (Yu does extend it to consider multiple constraint sets[91].)

Fuzzy Constraints and Objects

Constraints, like goals, should not be discrete; rather, they should describe a range of desirable restrictions. Fuzzy constraints should be expressible[39, 80].

Conditional Constraints and Objectives

> Constraints and objectives that depend on other conditions should be expressible. Constraints and objectives may depend on: (1) the design state, (2) agent state, (3) other agent state, or (4) other agent behavior.

While such a complete system remains future research, these features have been demonstrated individually. If available, such a constraint system would efficiently combine the three individual search methods (i.e., compromise, specialization, and generalization). Intuitively, I believe that numeric based methods, like MCSM, cannot effectively address such expressiveness. Future research will attempt to exploit simplex search notions in a qualitative search framework.

5.2.2. Specialization and Generalization

Recall that specialization derives more specialized relations, whereas generalization derived more general relation. Both methods are applied within the interactive search procedure. Hence, their depth and repetition of application are under user control. In the worst case, the resolution generation methods completely cover the representation space. However, its is more likely that some subset of the relations will be considered.

5.2.2.1. Analysis

Assume interactive search contains m conflicts involving p perspectives. Typically, specialization replaces some subset of relations, $n \mid n \leq m$, with s new specialized relations, $s > n$; however, s is typically equal to p which is often fewer than the possible s. The s new issues are found through either the relation or operator hierarchies.

For each application of specialization, the search space is limited by the branching factors of the operator and relation hierarchies. In the worst case where the root of the relation hierarchy is in conflict, all relations will be considered.

The analysis of generalization parallels that of specialization. Again, assume interactive search contains *m* conflicts involving *p* perspectives. Generalization adds some *g* relations. The *g* new relations are found through either the relation or operator hierarchies.

For each application of specialization, the search space is limited by the the branching factors of the operator and relation hierarchies. In the worst case where a leaf of the relation hierarchy is in conflict, all relations will be between it and the root will be considered. However, the relations suggested indirectly through the operator hierarchy must also be considered. In the worst case, one of them will add the root of the relation hierarchy, and again, all relations will be considered.

Finally, the interactive search method is not guaranteed to converge nor avoid local minima; neither are under program control. Given conflicts, a user is presented with nondominated compromises. Most decision support systems end search here; however, Oz gives the user specialization and generalization. Both introduce new relations.Since they reformulate the conflict, they may move out of local minima. Using the three methods, the user explores the search space and the group's contextual preferences. Convergence emerges as the user cycles through several alternatives, and eventually narrows the choice to one.

5.2.2.2. Related Dissolution Research

Generating resolutions is a creative task. Oz advocates an issue expansion approach. It attempts to generate alternatives which dissolve conflict. In contrast, planning systems typically use plan modification.

Conflicts are dissolved through plan modification, resource expansion, and bridging. Plan modification removes conflicts by considering different objects, operators, and their relationships. Some planners consider the availability of resources, but they typically do not dissolve conflicts by resource expansion[88]. This is mainly because resource expansion is expensive. However, in design, where resources are being defined, resource expansion is a good method of dissolution. Bridging may also be expensive; moreover, it is difficult to derive.

Bridging is type of replanning in that agent goals are reconsidered. Bridging attempts to dissolve conflict or reduce it by creating: (1) new relationships or (2) new technology. For example, time conflicts can be resolved by alteration or contingent sequence; consumable resource conflicts can be resolved by inventing renewable resources. The key difference between bridging and replanning is their focus: replanning resolves conflicts of an agent, while bridging resolves conflicts between agents. Using the planning paradigm, multi-agent conflict can be resolved by aggregating agent goals and then creating a plan[32]; bridging takes existing plans and repairs their negative interference. The Oz mechanism is a combination of the two. It aggregates goals to derive an integrated plan; however, it analyzes plans to derive new compatible goals. I speculate that for combining large plans with relatively little interference, bridging may be more efficient. Moreover, this is a common method of negotiators[66]. Also, typical planners are unable to resolve goal conflicts (e.g., ACHIEVE(X) & ACHIEVE(¬X)) which result from goal aggregation. There can also be methodological reasons for independent plan formation (chapter III).

Bridging, like planning, uses interference analysis to guide dissolution. For example, time multiplexing users of a resource falls naturally from the mutual consumption of relation instances[3]; relation instances can be used to describe resource usage over multiple states. Hence, planning methods can produce bridging relations. However, inventing new operators (technology) is not a typical planning method.

Sometimes, when multiple agents describe behaviors, objects or processes interfere. One method of resolving such conflicts is to substitute the incompatible entities with a new subsuming entity. For example consider two alternative proposals for library record keeping: (1) a manual card file, and (2) a computer database; due to costs, both systems cannot be used. One dissolution is to include both and expand the money supply (or alternatively reduce the costs). A more appealing solution is to create a new system satisfying the desired attributes of both systems. For example, the computer system may be desired for speed, remote access, and archiving; the manual system may be desired for familiarity, simplicity, and reduced eye strain. A solution may be a modified computer system: its operations use a manual system metaphor for familiarity and simplicity; its monitor is high resolution with large fonts for reduced eye strain.

New technology dissolution is accomplished by decoupling and combining the desired attributes. Zwicky has used this technique to predict new technologies[41]. Lenat combined this technique with pruning heuristics to discover new game strategies and math concepts[51-53]. It works best when: (1) the representational forms used are structurally similar to the abstract concepts, (2) there is a small set of combining operators, and (3) humans aid the concept evaluation. However, this method only finds concepts already defined implicitly in the representation language[54]. Yet, the method is effective in that it efficiently produces interesting concepts for human consideration. In the search for resolutions, this is a valuable technique.

When a conflict is similar to one previously considered, stored resolutions can help. Previous cases which have resulted in a satisfactory resolution, as well as those which resulted in an impasse, are useful. Successful cases can be instantiated and modified for the current context. Unsuccessful cases can prune resolutions from consideration. A case-base serves as a cache of unenunciated negotiation knowledge. It can contain resolutions which combine or individually embody dissolution, compensation, and compromise negotiation. However, case-based resolution can limit solution creativity. Reliance on cased-based resolution limits the creation of new resolution prototypes; it is an example of the *anchor and adjust* judgmental bias[5]. To provide for creative resolutions, case-based resolution systems must include a synthesis component.

5.2.2.3. Related Compensation Research

Most planning systems do not consider compensation because either: (1) they do not consider multiple agents, or (2) they do not consider a significant number of agent goals. Clearly, in multiple agent negotiations, one agent can be compensated by another. However, agents with complex goal structures can also try to compensate for their own unsatisfied goals; for example, a working agent unsatisfied with its salary can plan to gain other benefits.

Compensation depends on the goal structure of the "losing" agents. To determine how to compensate an agent, one can use a model of its goals. Previously, goal modeling has been used to construct persuasive arguments. Such arguments use the same tradeoff analysis used to determine compensation.

Sycara used graphs like that in figure 48 to construct persuasive arguments[83]. For example, if employees pushed for higher wages, PERSUADER would show that the company would have to reduce employment to keep labor costs down. Figure 47 illustrates how labor expenses are increased by both employment and economic increases. The company believes that to maintain market position, labor costs must be kept down; any increase in economic costs must be offset by a decrease in employment. PERSUADER uses a union belief model (figure 48) to understand that the union wants to increase employment; hence, the argument will influence the union since it points out the loss of desired goals.

PERSUADER uses goal relationships to construct persuasive arguments; it shows how an agent may respond to change based on its goal beliefs. An agent's goal beliefs can also be used to determine compensation. For example, the company may not agree to better union wages or benefits, but it may agree to better seniority wage and promotion policies. Compensation can be found by increasing satisfaction of sibling goals (NON-ECONOMIC of figure 48). When this fails, more general compensation may be had by moving up the union hierarchy to consider higher goals (e.g., UNION-SECURITY). By modeling agent beliefs of goals and their interactions, one can derive compensation from specific to general.

Oz's operator, object, and relation hierarchies are analogous to Sycra's goal belief tree.These hierarchies are searched for compensation like Sycra searches her goal tree for persuasive arguments. Sycara's trees are manually constructed; this allows them to accurately reflect the

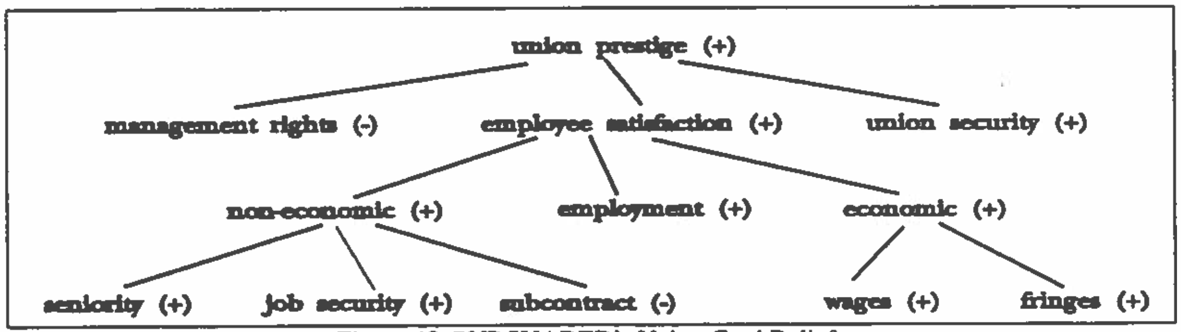


Figure 48. PURSUADER's Union Goal Beliefs.

designer intent. Oz''s hierarchies are automatically constructed; this allows them to accurately reflect the functional relationships represented in the system.

Sycara's trees can be viewed as an abstract summary of the operator hierarchies. Such simplification focuses the generation of arguments (or compensations), but implicitly excludes intermediate goals and more concrete goals. Automated hierarchies support greater of depth and breadth in abstractions. Large hierarchies may require us to skip some abstractions during compensation to prevent thrashing in minute details. Such compensation search control is still future research.

Oz, like Sycara's PERSUADER, has analytic and heuristic components[82]; however, the analytic component is based on process-oriented decision theory rather than strictly multiple attribute utility theory[9]. Oz's heuristic component is divided into dissolution and compensation, whereas these are implicitly combined in Sycara's case-base. Like Sycara, Klein uses a cased-based approach[47].

The above view of negotiation contrasts with DAI's view. For the most part, DAI views negotiation as a coordination problem. Message contents are simple task requests and accepts; the contract net illustrates this[18]. However, "multistage negotiation" illustrates how DAI negotiation can make more of the decision process collaborative[16]. Similarly, Werkman emphasizes message passing to engage in collaborative group resolution search[86].

This dissertation contributes to AI by applying and automating process-oriented decision theory for group design. The general search framework is not entirely novel; it is derived from Zeleny's IDEA. However, the use of specialization and generation is novel. Moreover, the domain model is a novel combination of decision theory and abstract planning[2].

6. Summary

Robust conflict analysis combines syntactic difference analysis with goal relationship knowledge[21, 29, 30]. Design language level analysis serves as a general mechanism for conflict detection, while goal knowledge serves to (re)characterize conflicts. Such characterization can be used to resolve conflicts and avoid conflicting behaviors. The two process are interdependent: goal interaction knowledge must be applied to goal conflicts; conversely, design conflicts must be detected to be resolved.

Determining the linkage between behaviors and goals is a problem. Resolving negative interactions is another. Most systems rely on basic plan modifications: (1) alternative operators, (2) reordering, (3) reestablishment, and (4) alternative unification. Few attempt to modify the underlying causes of the conflict; for example, resource shortages, or unrealistic expectations. When plan modifications fail, modification of environmental or agent constraints can be helpful.

The conflict detection method combines a top-down model of goal interactions with simple language level analysis. Conflict resolution is achieved through plan and goal modification. My research goal is not to introduce new plan modification operators, but to combine such operators with design and goal modification in a negotiation framework.

CHAPTER VI

A DETAILED EXAMPLE

The representations and algorithms presented in this dissertation provide negotiation aid for designers. Additionally, records of their use provide rationale for the design. This chapter demonstrates the sufficiency of the methods to generate useful resolutions and aptly record negotiated decisions. It is done so by rederiving a portion of a simple library system. In fact, the library system is rederived twice. First, to demonstrate that **Oz** can derive a negotiated design described in a case study. Second, to demonstrate that **Oz** can suggest still more resolutions. In the end, I conclude that the representations and algorithms do support negotiated design. However, before the derivations, the library case study is presented.

1. The Library Problem

In 1968, Burkhalter and Race analyzed the charge-out (loan) period for the University of Michigan General Library[10]. Here is how they perceived their problem.

> Library administration asked, "Should the length of the two-week charge-out period for student book loans be changed?" Initially, the question arose in response to increasing pressure on the circulation staff. Lengthening the charge-out period might reduce the time the staff must devote to activities such as processing overdues and renewals. On the other hand, a shortening of the charge-out period might lead to greater book availability for the patrons. The purpose of this study is to identify the factors affected by a change in the charge-out period which lead to a change in cost or patron service, and then to attempt to assess the relative degree of change so that a decision can be made as to the optimal charge-out period.— p. 11[10]

To begin their analysis, they described a portion of the initial library state: (1) a loan period of two weeks, (2) renewal at the desk, and (3) overdue notices every week for three weeks. Next, they analyzed the cost and effectiveness of this arrangement, with particular attention to renewal and overdue notice procedures. Finally, they redesigned the library to include: (1) a loan period of three weeks, and (2) overdue notices every week for two weeks.[1]

This study appears as an example of the type of implicit negotiations that occur during design. As is typical, a central designer (Burkhalter and Race) analyze group needs, and then implements a solution. However, the varied views of the group, and their negotiations, go unrepresented. In contrast, Oz explicitly represents and reasons about stakeholder perspectives and their negotiations. To represent the library perspectives for this case study, I have had to infer the original rationale.

I have rationalized the library design as a negotiation between the librarians' group and the patrons' group. This is common in library design[55,57]. Decisions are often described as trade-offs between the goals of a representative librarian and a representative patron. Additionally, Burkhalter and Race explicitly referred to such trade-offs. When they did, I formally represented them explicitly and accurately. Hence, I believe that if they had the same representations to use, they would have derived perspectives similar to those that follow.

Before presenting the two library perspectives, the results of the Burkhalter and Race's analysis are described. From user surveys and analysis of circulation patterns, they derived tables and graphs describing the initial library conditions. These analyses are represented in Oz. They are

[1]Burkhalter and Race did also consider some subprocesses of these operations. For example, they analyzed the cost effectiveness of the notice copy mechanism. Their analysis revealed that Xerox copying of notices would be more efficient than their current ditto procedure. Hence, the modified library implemented this change. However, I did not conduct the study at this level of detail. Since, Burkhalter and Race mainly focused on loan period and overdue, so did I.

used to guide resolution search. Next, their analysis of the library's loan period and overdue notice procedures are summarized.

1.1. Loan Period

Burkhalter and Race divided loan period into two parts: useful time and idle time. During use, patrons actually avail themselves of their resources. However, the time before resource return and after resource use is idle time. Librarians explicitly have the goals of increasing resource usage while reducing idle time. In an ideal world, upon completion of usage, each patron would immediately return their resources. Instead, resources are often idle before their return. Librarians can attempt to control this situation through loan period duration. By reducing loan period duration, patrons are encouraged to at least renew resources, if not return them, after a period of time. Loan period determines the time frame, while late fees determine the amount of encouragement. By analyzing renewal patterns and surveying patrons, Burkhalter and Race described how resource usage varied with loan duration.

Figure 49 illustrates the representation of Burkhalter an Race's usage graphs. I have combined their graduate and undergraduate graphs into a single one since they did not consider distinct loan periods for graduates and undergraduates. Also, unlike the graph, theirs are smooth non-linear curves. In contrast, Oz graphs are formally represented by a set of linear constraints. Such linearity allows us to apply linear programming methods to analyze the graphs.

Figure 49 shows how student resource usage varies with time. In the first few days, student's only partially complete their use. However, after 20 days, students are nearly completely finished using resources. According to the graph, loan periods longer than 20 days will encourage resource idle time. However, such graphs must be interpreted in proper context. The 14 day loan

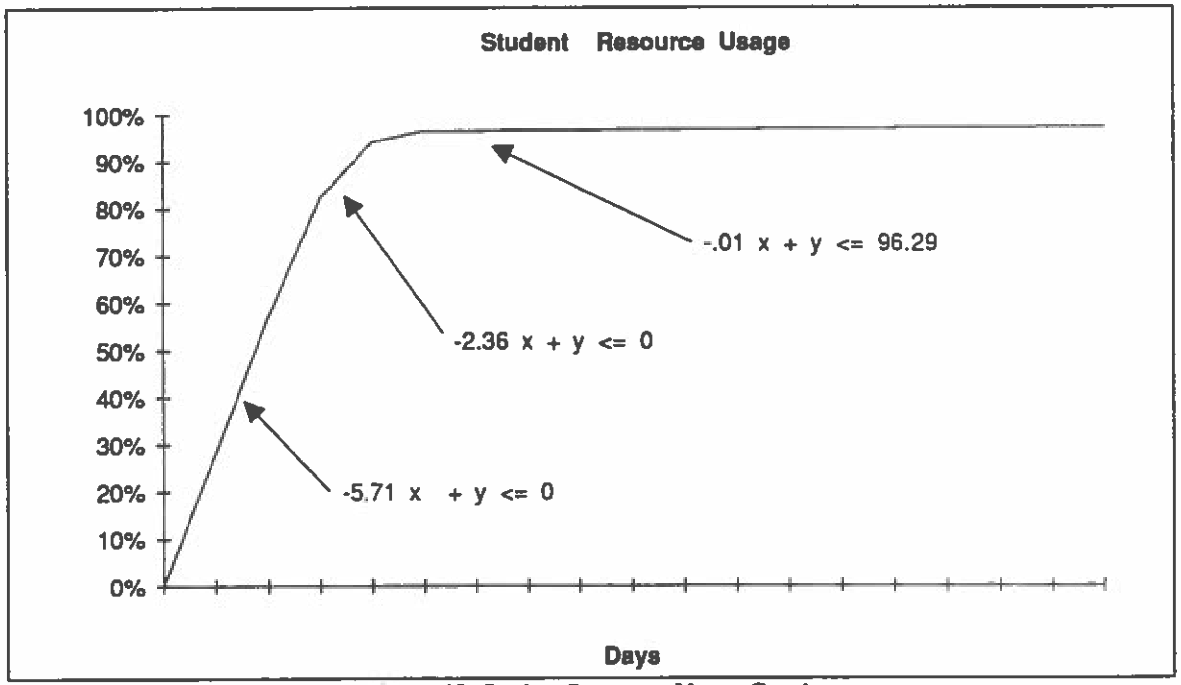


Figure 49. Student Resource Usage Graph.

period coupled with the 7 day overdue notice may have caused students to return resources near 21 days. On the other hand, the user surveys indicate that students would return resources after 21 days even with a longer loan period. That is, 21 days is often sufficient for student borrowing.

As Burkhalter and Race state, the circulation work load was one of the factors which initiated their case study. Circulation staff were being overrun by patron renewal requests. Hence, longer loan periods were posed as a means to reduce renewal requests. While Burkhalter and Race did not explicitly represent reduced renewal cost as a function of loan period duration, I have. Figure 50 shows how I interpret their perception of renewal cost. The main element of the figure is simply that cost decreases as loan period duration increases. Eventually, (around 45 day by my estimate) renewal costs are negligible.

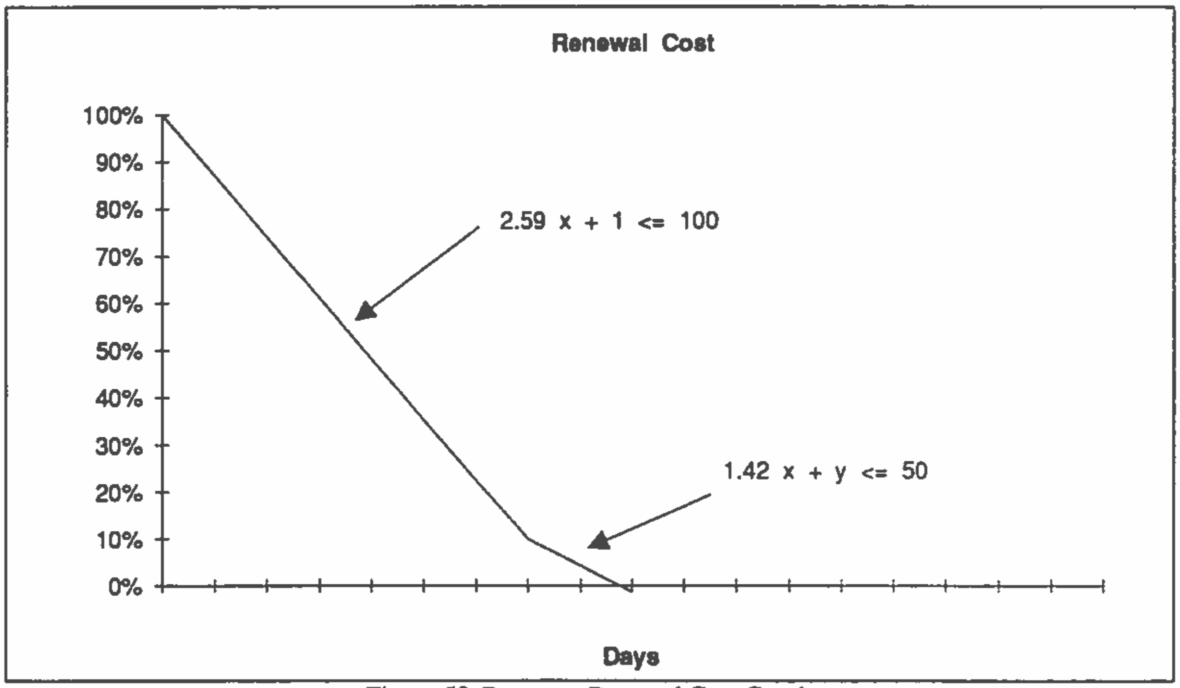


Figure 50. Resource Renewal Cost Graph.

1.2. Overdue Notice

Burkhalter and Race applied the same sort of analysis to overdue notices. Librarians desire to aid patrons in remembering to return their resources. However, three other issues mitigate this goal: (1) costs, (2) responsibility, and (3) availability. Burkhalter and Race measured the effectiveness and cost of the overdue notices. (Initially, three notices were sent: one every week after a

Notice	1	2	3
Number sent	1600	750	530
Resulting returns	850	220	30
Percent effectiveness	53%	29%	6%
Percent of total overdues returned	54%	14%	2%
Relative cost per book	100	181	940
Percent of total overdue notice cost	56%	26%	18%

Figure 51. Table of Overdue Notice Evaluation.

resource was due.). Figure 51 shows a table of their results. They concluded that:

> ...the second notices entail one-fourth of the cost while producing less than one-seventh of the returns; and the third notices entail one-sixth of the cost while producing one fiftieth of the returns. This low return for the dollar, coupled with the philosophy that it is the patron's responsibility to return books, led to the recommendation that the third notice should be considered for elimination.— p. 20[10].

Figures 52 and 53 show the graphs for overdue costs and effectiveness. The graphs illustrate the reduction of costs and effectiveness associated with later notices.

Burkhalter and Race use their analysis to rationalize the reduction of overdue notices from three to two. Similarly, Oz uses the analysis in its perspectives to model attribute relationships. Such relations are useful in constraining the space of resolutions and depicting stakeholder preferences. Next, two perspectives based on the above analysis are presented.

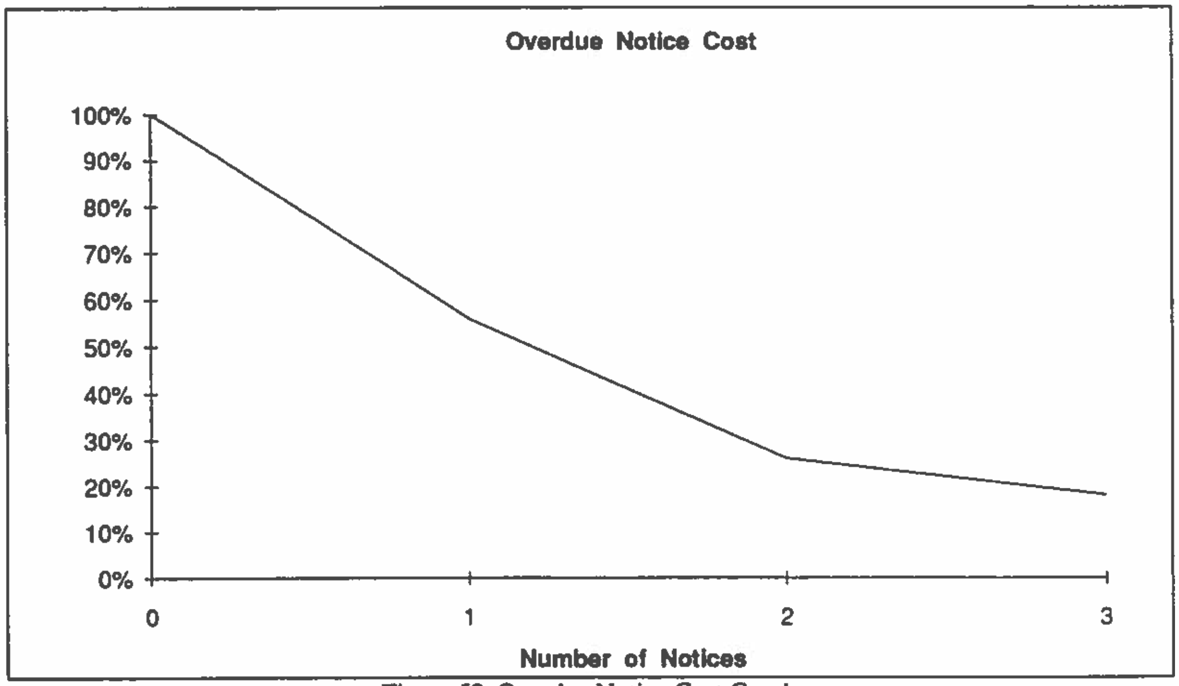


Figure 52. Overdue Notice Cost Graph.

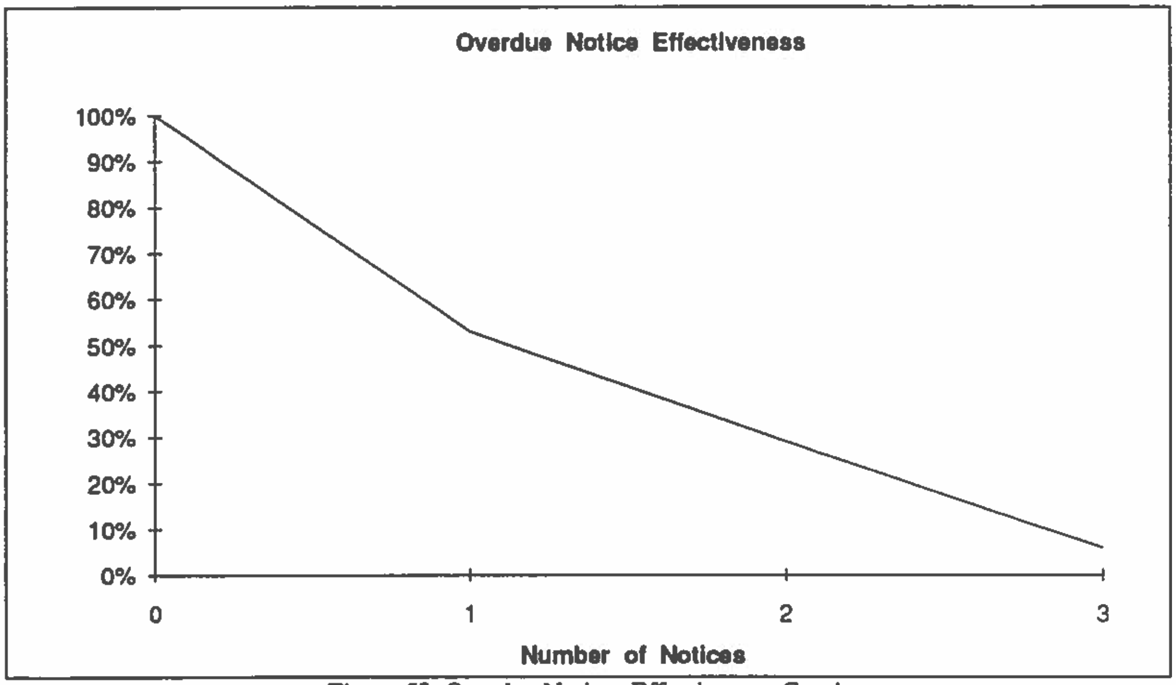


Figure 53. Overdue Notices Effectiveness Graph.

1.3. Two Perspectives

The librarian and patron perspectives are quite similar. They both modify the generic library model by attaching preferences associated with loan period and overdue notices. The both use the Burkhalter and Race's analysis of loan period usage and cost, and overdue notice effectiveness and cost. However, they differ in their preferences. The librarian is focused on reducing costs while maintaining high resource usage. In contrast, the patron only cares about long loan periods and multiple overdue notices. Figures 54 and 55 illustrate the librarian and patron perspectives. (Note, they only contain preference modifications to the library model. See Appendix A the complete library model, including the initial relation instances.)

Figure 55 shows the librarian's perspective. It represents the librarian's desire to: (1) loan student resources (ON_LOAN), (2) notify students of overdue resources (GIVE_NOTICE), and (3) allows students to access their loan records (KNOWS_RECORDS). These are only a few of the many

```
(in-perspective 'Librarian)

(def_establish
  (on_loan(student1,resource1,loan_period1))
  (knows_records(student1,student1,resource1))
  (give_notice(library1,student1,overdue_notice1)))

(def_mod_object "loan_period"
 :atts '(
       ("Duration" :val 14 :min 0 :max 365 :obj min)
       ("Usage" :val 100 :min 0 :max 100 :obj max)
       ("RenewCost" :val 0 :min 0 :max 100 :obj min)
       )
 :att_constraints '(
                 ;; Usage
                 (-5.71 1 0 <= 0)
                 (-2.36 1 0 <= 47)
                 (-0.01 1 0 <= 96.29)
                 ;; RenewCost
                 (2.5882 0 1 >= 100)
                 (1.1429 0 1 >= 50)))

(def_mod_otyp "overdue_notice"
 :atts '(
        ("Number" :val 3 :min 0 :max 5 :obj min)
        ("Cost" :val 100 :min 0 :max 100 :obj min)
        ("%Wasted Effort" :val 100 :min 0 :max 100 :obj min)
        )
 :att_constraints '(
                    ;; Cost
                      (44 1 >= 100)
                      (8 1 0 >= 42)
                    ;; Wasted Effort
                      (-2 0 1 >= 0)
                      (-23 0 1 >= -40)
                      (-47 0 1 >= -135)))
```

Figure 54. The Librarian's Perspective.

goals associated with a library system.

My focus on loaning and notifying reflects Burkhalter and Race's case study. The addition of the access goal is to demonstrate that the following negotiation can be part of a larger design effort. Since the access goal is shared (i.e., the same) for both the librarian and the patron, it will be determined not to be in conflict, and hence, not part of the negotiations. This will demonstrate how the integration methods only focus on the conflicting parts of the design, and allow the other parts to be combined without negotiation.

In addition to goals, figure 54 also shows the preferences of the librarian. The librarian desires to maximize resource usage, yet maintain resource availability. Decreases in loan period duration increase resource availability. Hence, the librarian's preferences are represented as maximizing usage, minimizing duration, and also minimizing renewal cost. The relationship between duration, usage, and cost as depicted in the previous section is represented by the attribute constraints. Additionally, the librarian's overdue notices are represented. The librarian prefers to minimize costs associated with the notices, while maximizing their effectiveness. Rather than model effectiveness, I choose to model wasted effort—the inverse of effectiveness. Hence, the librarian prefers to minimize wasted effort. These preferences and the associated relationship between them and overdue number (from the previous section) are represented as the modification of the OVERDUE_NOTICE object.

Figure 54 also illustrates how Oz can derive a design using specific values for attributes, while having seemingly conflicting preferences. In the figure, the librarian has the objectives of minimizing loan period duration and overdue notice number, yet the values are set at 14 and 3, respectively. This mechanism is used to allow the derivation of a previously negotiated design. For example, in the Burkhalter and Race case study, their librarian's own availability preferences would suggest minimal duration, while their librarian's (and student's) usage preferences suggest longer duration. Apparently, previously they negotiated a 14 day duration and three overdue notices. Rather than rederive that earlier negotiation, I set up the librarian's perspective to derive the initial library described in Burkhalter and Race's case study, and then integrate that design with a conflicting design derived from the patron's perspective.

Figure 55 illustrates the patron perspective. It is nearly identical to that of the librarian. Like the librarian, the patron desires that the library: (1) loan student resources (ON_LOAN), (2) notify students of overdue resources (GIVE_NOTICE), and (3) allow students to access their loan records (KNOWS_RECORDS). Additionally, the patron desires to maximize resource usage, and unlike the library, maximize loan period duration. Moreover, the patron does not care about library renewal costs. Figure 55 also represents the patron's overdue notice preferences. The patron prefers to maximize the number of notices. Unlike the librarian, the patron does not care about notice costs. Again, the associated relationship between attributes are represented as linear constraints.

```
(in-perspective 'Patron)

(def_establish
  (on_loan(student1,resource1,loan_period1))
  (knows_records(student1,student1,resource1))
  (give_notice(library1,student1,overdue_notice1)))

(def_mod_object "loan_period"
 :atts '(
       ;;While patron's only care about usage.
       ;;All preferences are included so constraints can reference them.
       ("Usage" :val 100 :min 0 :max 100 :obj max)
       ("Duration" :val 365 :min 0 :max 365 :obj nil)
       ("RenewCost" :val 0 :min 0 :max 100 :obj nil)
       )
 :att_constraints '(
                  ;; Usage
                  (-5.71 1 0 <= 0)
                  (-2.36 1 0 <= 47)
                  (-0.01 1 0 <= 96.29)
                  ;; RenewCost
                  (2.5882 0 1 >= 100)
                  (1.1429 0 1 >= 50)))

(def_mod_object "overdue_notice"
    :atts '(
         ("Number" :val 5 :min 0 :max 5 :obj max)
         ;; Patron's don't care about effectiveness or cost
         )
    :att_constraints '(
                     ;; Cost
                     (44 1 >= 100)
                     (8 1 0 >= 42)
                     ;; Wasted Effort
                     (-2 0 1 >= 0)
                     (-23 0 1 >= -40)
                     (-47 0 1 >= -135)))
```
Figure 55. The Patron's Perspective.

1.4. Two Designs

Given the two perspectives, the planner is able to automatically derive a design achieving the goals. Figure 56 shows Oz's depiction of the librarian's goals and derived design. The design in the figure illustrates how the initial library conditions enable students to access their loan records after they have borrowed a resource. It also illustrates the BORROW and GIVE_NOTICE operators. The initial library conditions enable students to borrow resources and then receive overdue notices.

Notice that the design does not use abstract operators. To most closely simulate the case study, I have instructed the planner to use concrete operators. Hence, the operator GET_RESOURCE was specialized to BORROW to reflect the previous design. To derive a design from scratch, one would run the planner in abstract mode. Then, operators are specialized only if required by the design or preferences.

The patron's goals and design look identical to that depicted in figure 56. What is not shown are the differing preferences in the object hierarchies and the differing relation instances produced in the design. In each perspective, the objects do contain the preferences described above. Those preferences cause the planner to create instances of relations which have the appropriate attributes. Hence, while not shown in figure 56, the design for the librarian has a loan period duration of 14 days and an overdue notice number of 3. In contrast, the patron's design has a loan period duration of 365 days and an overdue notice number of 5.

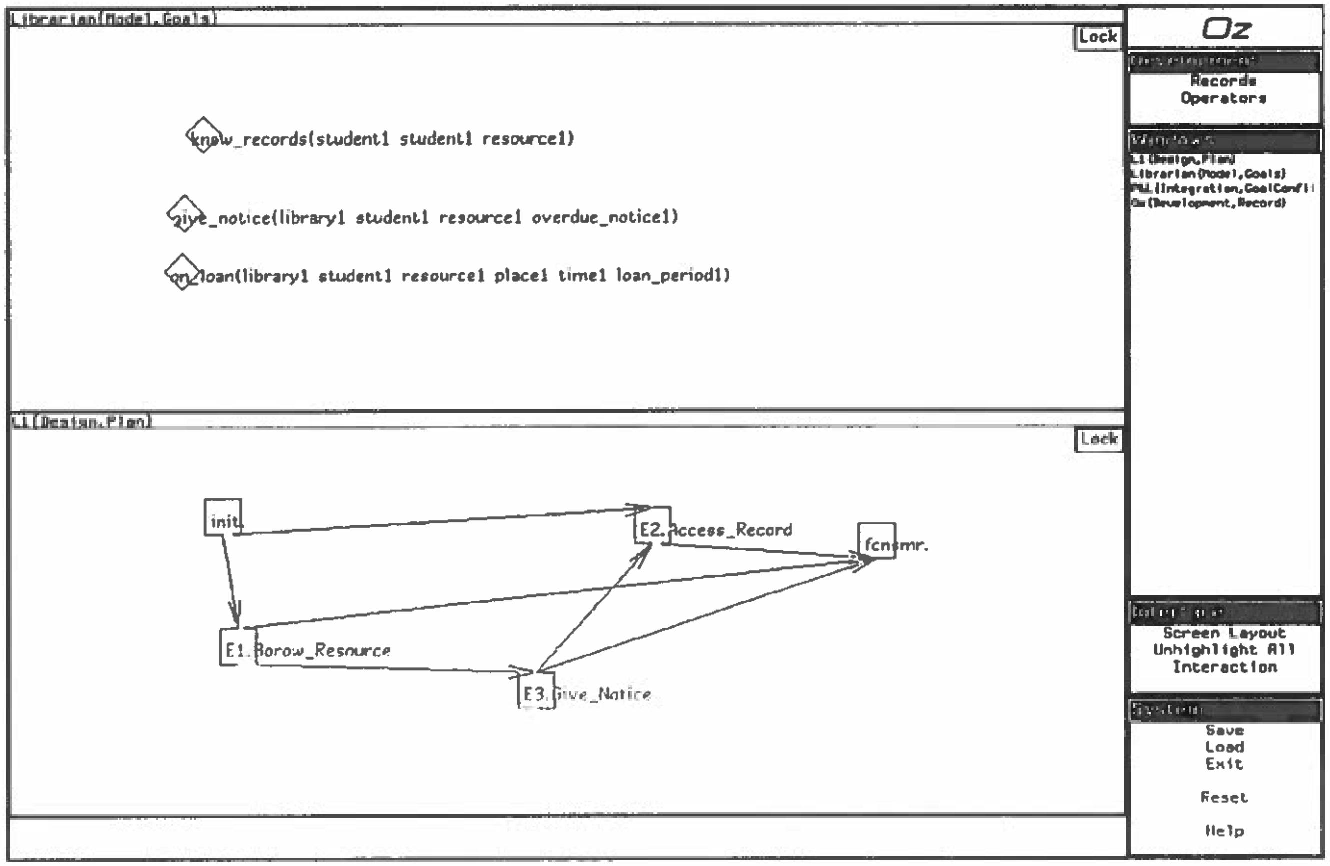


Figure 56. Oz Screen Depiction of the Librarian's Goals and Design.

1.5. Original Perceived Conflict

Based on their understanding of patron goals in general and analysis of the initial library in particular, Burkhalter and Race determined the following conflicts:

(1) Loan period duration. The initial loan period did not achieve the proper balance of availability, renewal cost, and usage. Specifically, the patron's desire for longer loan period duration was in conflict with the library's initial desire for a shorter loan period duration. This conflict was first noted after using the initial library system and observing the circulation staff spending too much time handling renewal requests.

(2) Overdue notice number. The initial overdue notice number did not achieve the proper balance of cost and effectiveness. Specifically, the patron's desire for more notices was in conflict with the library's initial desire for fewer notices. This conflict was first noted after attending to the loan period duration conflict. During their loan period analysis, Burkhalter and Race also considered overdue notices. From their analysis they uncovered that overdue notices were not as cost effective as desired.

Given these conflicts, Burkhalter and Race derived a resolution.

1.6. Original Resolution

Using their analysis, Burkhalter and Race may well have considered many alternative loan period and overdue notice resolutions. However, they may have been directly drawn to the 21 day alternative due to its prominence. It is: (1) near the top of the usage curve, and (2) has some reduction in cost. Similarly, they state (see section 1.2) they were drawn to a two notice policy given the extremely low effectiveness of the three notice policy. Hence, rather than consider all available alternatives, I speculate they were drawn to consider the *status quo*, or a 21 day loan period with a two overdue notices. After weighing the alternatives, they choose the 21 day student loan period and two overdue notices.

1.7. Original Rationale

Burkhalter and Race rationalized their resolution based on trade-offs between the patron and library goals. (See the quote in section 1.2.) First, they felt the increase in loan period duration was justified by: (1) reduced renewal costs, (2) increased resource usage, (3) increased satisfaction of patron preferences. They felt this despite the decrease in resource availability. Second, they felt the decrease in overdue notices was justified by reduced notice costs. They felt this despite the slight decrease in overall notice effectiveness and patron preferences to the contrary. Hence, the net effect on the *status quo* was to increase the effect of patron loan period preferences on the one had, while reducing the effect of patron overdue notice preferences on the other. Such reciprocity is common in negotiations.

2. The Derivations

The following two sections present derivations based on the Burkhalter and Race case study. In both sections, the design derived from the librarian's perspective is used to represent the initial library analyzed by Burkhalter and Race. However, the librarian's perspective represents more idealized (extreme) goals than the librarian's design implements, i.e., the duration goal is to be minimized while the design reflects the initial loan period duration of 14 days. On the other hand, the patron perspective represents idealized patron goals and a design implementing those goals, i.,e., the design has a 365 day loan period.

The following two derivations will detect the loan period and overdue notice conflicts, suggest alternative resolutions, and derive a new perspective and design reflecting the resolved goals. The first derivation will strictly follow the case study, while the second will present other resolutions. In each section, only directly relevant aspects of the algorithms are presented. Refer to chapter V for their complete description.

Figure 57 illustrates a trace of the library development. First, perspectives representing the library and patron are created (called Library and Patron). Next, designs are automatically created, thereby achieving the goals for each perspective. Next, the designs are integrated and their conflicts are noted (and resolved) in the integration record named P&L. Next, a perspective is automatically generated representing the resolved goals of the two perspectives. Finally, an integrated design the automatically derived.

To begin integration, an Oz user selects the *integrate* operator from the development popup menu, selects the designs, and then hits the go button. Each derivation description picks up the moment after the two designs have been selected for integration.

2.1. Strict Rederivation

In this derivation, Oz integrates the perspectives and designs of the previous sections. First, goal correspondences are identified and their surface conflicts are noted. Next, plan interference between the goals is identified. Then, the conflicts are presented to the arbiter. Next, compromises are generated and a resolution is selected. Finally, a new perspective and design are derived representing the negotiations.

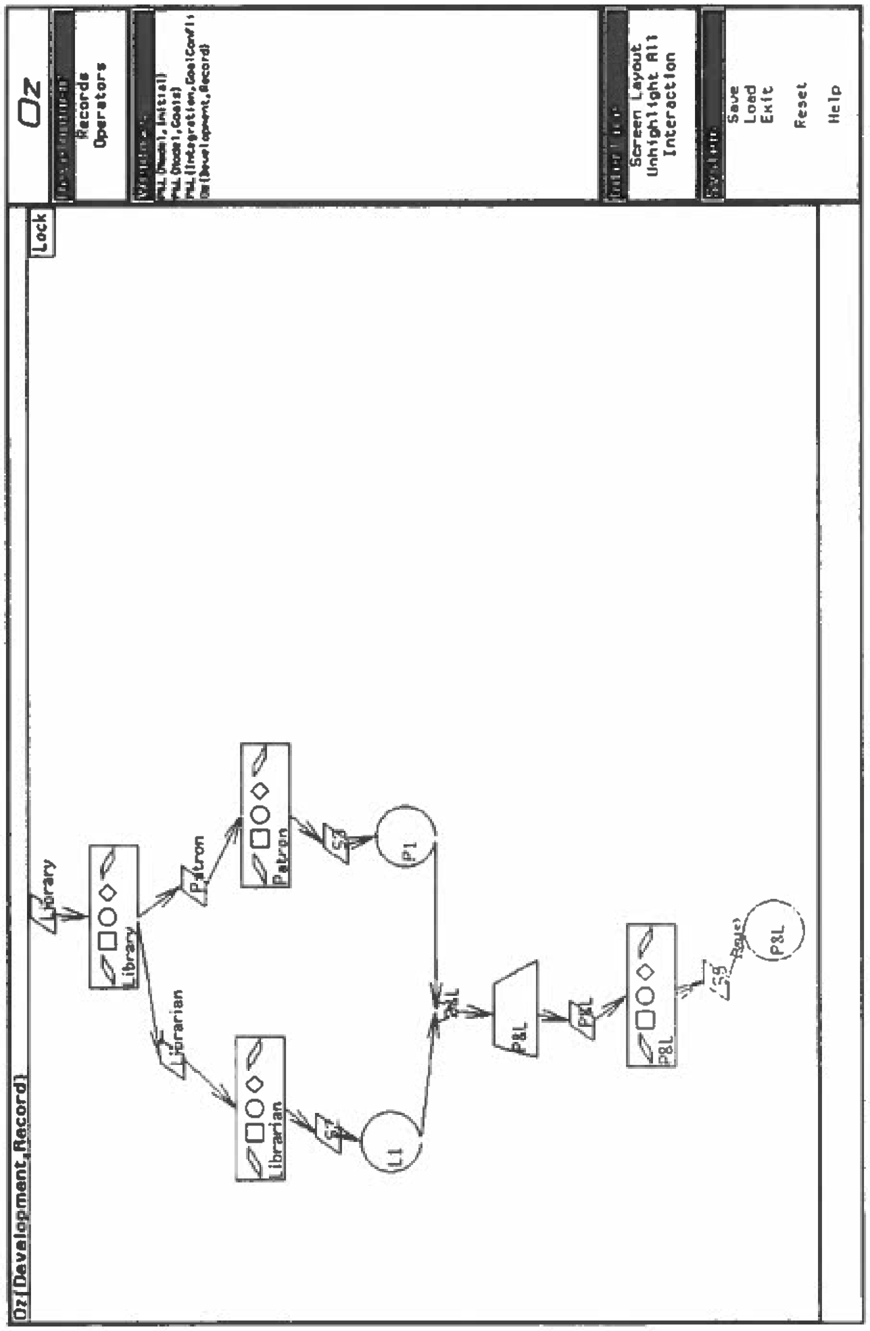


Figure 57. Oz Screen Depiction the Library Development.

2.1.1. Correspondences and Conflicts

Conflict detection begins by finding the best correspondences between the goals of the two perspectives. Since the two perspectives have identical goals, correspondence identification is simple. In this case, goals with the same names, but in different perspectives are marked as corresponding.

Any difference in the goals are noted as part of the matching process. In this case, differences are noted between the correspondences of the ON_LOAN, and GIVE_NOTICE, and KNOW_RECORDS goals. In the first two cases, differences are classified as object attribute conflicts. For the ON_LOAN goal, the value of the LOAN_PERIOD attribute DURATION is different. It is 365 in the patron perspective while 14 in the library perspective. For the GIVE_NOTICE goal, the value of the OVERDUE_NOTICE attribute NUMBER is different. It is 5 in the patron perspective while 3 in the library perspective. In contrast, the KNOW_RECORDS is identical; however, it is still marked as a possible means conflict. The interference procedure will determine if the two KNOW_RECORDS have interfering plans.

2.1.2. Interference

Interference detection begins by determining how a goal has been implemented in a plan. Next, it back propagates pre-conditions. Corresponding goals have their pre-conditions compared. If they delete (consume) each other's resources, then they are said to interfere. However, if they simply use the same relation instances, then they do not interfere.

2.1.2.1. Loan Period

The ON_LOAN goals of the two perspectives are implemented identically in their respective plans. In each plan, the operator BORROW_RESOURCE is the single operator that implements the goal. Figure 58 illustrates the borrow operator. Note that the only relation it deletes is the possession relation. However, back propagating the two ON_LOAN goals through their respective BORROW_RESOURCE operators leads to the same relation instance. Hence, these two goal implementations interfere.

When two goals are found to interfere, an attempt is made to replan to achieve them. Using the initial conditions and the conjoined goals (i.e., not considering interactions with other goals), the planner is applied to determine if the conflict can be resolved by replanning. If so, both goals may be included in the final plan if the arbitrator agrees that both goals and their replanned implementation are acceptable. Given the ON_LOAN conflict and the library operators, the planner cannot resolve the conflict through replanning.

2.1.2.2. Late Notice

The GIVE_NOTICE goals of the two perspectives are implemented identically in their respective plans. In each plan, the operators BORROW_RESOURCE and GIVE_NOTICE are the operators that

```
(def_operator "Borrow_Resource"
 :description "A loanable RESOURCE is LOANED to an AGENT for LOAN_PERIOD."
 :objects '(agent1 agent2 resource1 time1 time2 place1 loan_period1)
 :preconditions
 '((loan_period(agent1 resource1 place1 time1 loan_period1))
   (time_is(time2)))
 :deletes '((possess(agent2 resource1 place1 time1 loan_period1)))
 :adds '((possess(agent1 resource1 place1 time2 loan_period1))
         (on_loan(agent2 agent1 resource1 place1 time2 loan_period1))))
```

Figure 58. A Library BORROW Operator.

```
(def_operator "Give_Notice"
    :description "The LIBRARY gives PATRONs OVERDUE NOTICEs for RESOURCEs."
    :objects
      '(library1 patron1 resource1 place1 time1 loan_period1 overdue_notice1)
    :preconditions
          '((on_loan(library1 patron1 resource1 place1 time1 loan_period1)))
    :deletes '()
    :adds '((give_notice(library1 patron1 resource1 overdue_notice1))))
```

Figure 59. A Library GIVE_NOTICE Operator.

implement the goal. Figure 59 illustrates the notice operator. Note that it does not delete any relation. Hence, these two goal implementations do not interfere through the GIVE_NOTICE operator; however, they do interfere through the BORROW_RESOURCE operator. Again, the goal is back propagated through all operators implementing it, and then those consuming pre-conditions are compared for conflict. Hence, the GIVE_NOTICE goals do interfere. Given the GIVE_NOTICE conflict and the library operators, the planner cannot resolve the conflict through replanning.

2.1.2.3. Record Access

The KNOW_RECORDS goals of the two perspectives are implemented identically in their respective plans. In each plan, the operator ACCESS_RECORD is the single operator that implements the goal.[2] Figure 60 illustrates the access operator. Note that it does not delete any relation.

```
(def_operator "Access_Record"
    :description "An AGENT can read the library loan records."
    :objects '(agent1 agent2 resource1)
    :nots '((secure_records(agent1 agent2 resource1)))
    :preconditions '((access_records(agent1 agent2 resource1)))
    :deletes '()
    :adds '((know_records(agent1 agent2 resource1))))
```

Figure 60. A Library ACCESS Operator.

[2]One may notice the directed link between GIVE_NOTICE and ACCESS_RECORD in the plan of figure 57. That link does not indicate a pre-condition dependency as can be seen if figure 60. Instead, it is simply OPIE's way of indicating a *possible* ordering for unordered operators.

Hence, these two goal implementations do not interfere.

Note there is a distinction between goal conflict and goal interference. Goals which conflict achieve different states. Goals which interfere use each other's resources. Goals which conflict may or may not interfere. However, in either case, one must negotiate. If they interfere, one can negotiate about which goal will be included into the design. If they do not interfere, one can negotiate about whether one wants to include both achieved states in the design.

When two goals do not interfere, no replanning effort is made. Hence, the planner is not invoked to attempt to resolve the ACCESS_RECORD "conflict".

2.1.3. Initial Issues

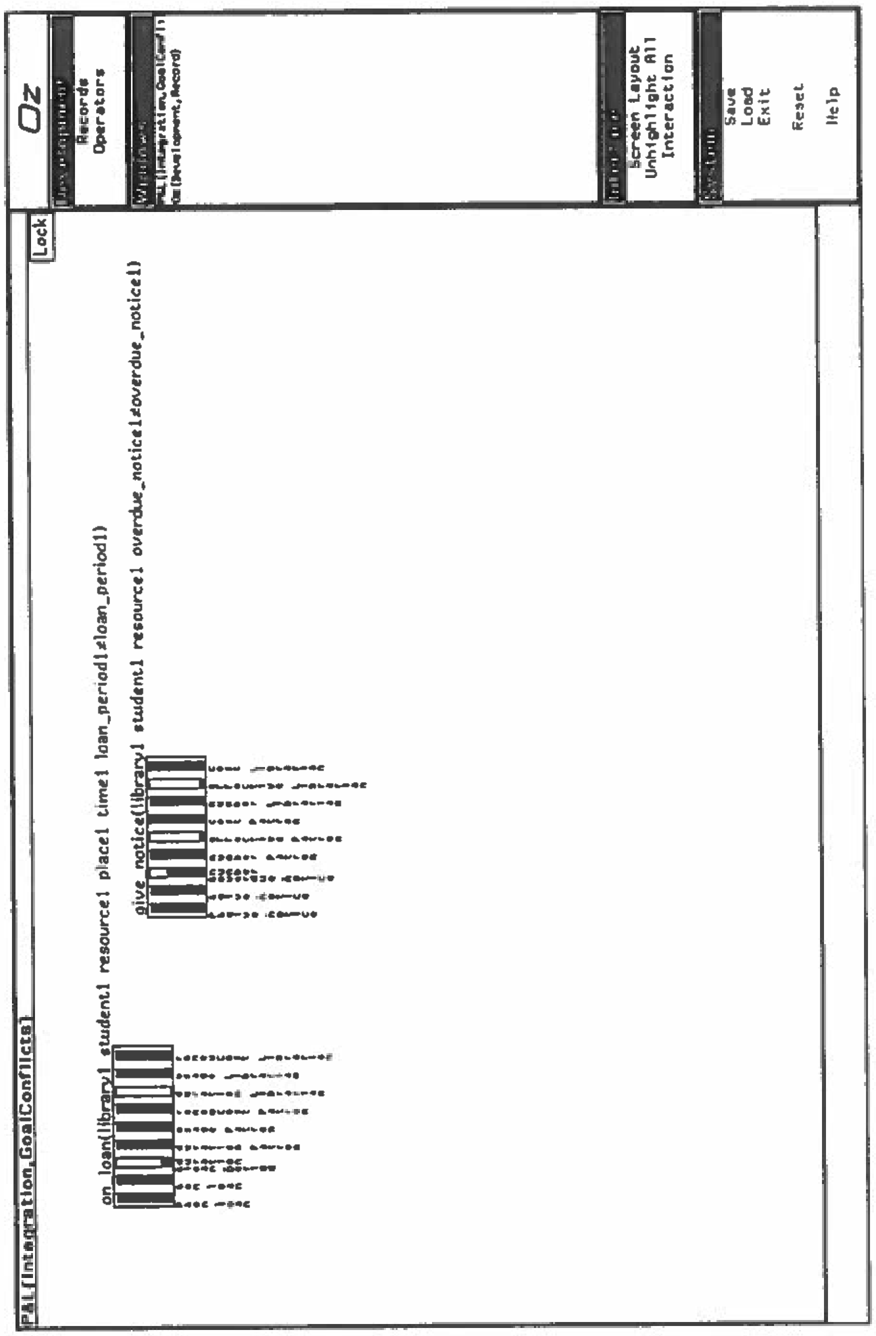
Figure 61 shows an Oz depiction of the initial issues. The relations ON_LOAN and GIVE_NOTICE are shown with special attention to the conflicting objects. For example, the ON_LOAN conflict is displayed as:

```
on_loan(library1 student1 resource1 place1 time1 loan_period1≠loan_period1)
```

This indicates that the corresponding objects LOAN_PERIOD1 from the two perspectives are in conflict. (In this case, the two perspectives use the same name; however, it is possible for goals to match even though they use different object names.)[3]

Preference satisfaction is displayed below the relations. In each case, preferences are displayed in the following left to right order: relation preferences, relation attribute preferences, and object attribute preferences (if they exist). For each of these preference types, first the average of

[3]In the case of conflicting relations or relation attributes, the relations would be displayed as conflicting. For example, GIVE_NOTICE≠GIVE_NOTICE. Refer to Chapter V, section 1.3).



P&L(Integration,GoalConflicts)

on_loan(library1 student1 resource1 place1 time1 loan_period1≠loan_period1)

give_notice(library1 student1 resource1 overdue_notice1≠overdue_notice1)

Figure 61. Oz Initial Screen Depiction of the Goal Conflicts.

the perspectives is displayed, and then preferences from each perspective is displayed (if they exist).

Reading from right to left, consider the preferences associated with ON_LOAN. The three rightmost are the librarian's object attribute preferences for LOAN_PERIOD1. The loan period attribute preferences are: resource cost, usage, and duration. The next three preferences toward the left are the patron's object attribute preferences for LOAN_PERIOD1. Again, the loan period attribute preferences are: resource cost, usage, and duration. The next preference indicates the average satisfaction of all the LOAN_PERIOD attribute preferences, i.e., resource cost, usage, and duration from both perspectives. This averaged preference is named &LOAN_PERIOD DURATION. Loan period is the object and duration is the attribute whose value differed in the two designs. The next preference to the left is the averaged ON_LOAN attribute preference. Since there are no attribute preferences associated with the ON_LOAN relation, no individual preferences are displayed. However, notice that its satisfaction is at 100 percent. Whenever there is no preferred ordering associated with a range, its satisfaction is displayed at 100 percent, thereby allowing the arbitrator to effectively ignore its contribution to an alternative's overall achievement. Finally, the last preference to the left is the averaged ON_LOAN relation preference. Since there are no relation preferences associated with ON_LOAN, no individual preferences are displayed. The GIVE_NOTICE preferences are displayed in similar fashion.

Each displayed relation represents an alternative. Hence, an initial relation represents two concepts. It represents the initial conflict, where the display notes the conflicting components with the ≠ symbol. And, it represents an alternative where the conflict is resolved. In the case of the initial relation, this conflict is resolved by choosing the values from the first perspective given to the integration; the value from the patron's perspective. Hence, the first relations displays the conflicts and the alternatives where the first perspective's values are chosen. Subsequent alternatives will be generated and displayed using the same format. However, they will only represent resolutions; hence, no conflicts will be displayed.

For each relation (alternative) displayed, preference satisfaction is displayed in the form of a bar graph. The more satisfied a preference, the higher the bar graph. These first relations display the alternative of a 365 day loan period duration and 5 overdue notices. Hence, one can see from the initial relations displayed, preferences from the two perspectives concerning the patron's proposed values. Thus, the patron's preferences are higher (equal or) than the librarian's for each relation.

From these initial issues, the arbitrator considers the preferences of the two perspectives and considers whether to: (1) accept the initial alternative, i.e., the values from the patron perspective, or (2) apply a resolution method. For this case study, the arbitrator applies the compromise method.

2.1.4. Generated Compromises

The arbitrator applies the compromise method to each of the conflicts in turn. First, loan period duration compromises are generated. Next, late notice number compromises are generated. Such compromises are generated by simply choosing the compromise method from a popup menu and then clicking on one of the relations. In this way, the three methods can be interleaved. That is, first compromises can be generated. Then, those relations can be generalized or specialized. Such an approach is shown in the expanded example. However, here only compromise is applied.

2.1.4.1. Loan Period

Figure 62 shows an Oz depiction of the loan period duration compromises. The compromises are linked to the to initial relation via an icon depicting the compromise operation.

Figure 63 shows the table of compromise solutions depicted in figure 62. Each row indicates a resolution. The first column lists the duration values for loan period. The next two indicate the derived resource usage and cost associated with a particular duration. To determine the satisfaction a perspective derives from such resolutions, one must consider the goals. For example, the resolution of zero duration, zero usage, and 100 percent cost is an extreme resolution which only satisfies the librarian's goal to minimize loan period duration (to achieve resource availability). In contrast, the resolution (20.9, 96.5, 45.7) appears to be a more balance nondominated resolution. It nearly achieves 100 percent satisfaction of resource usage and has lower renewal costs, which are both important to both perspectives. On the other hand, it achieves some balance between a short and long loan period duration. If this resolution were chosen (as it was), it would indicate that the librarian's duration preference is regarded more important than the patron's opposing preference. Yet, one preference was not simply chosen over the other. Instead, their preferences were combined.[4]

The resolutions of figure 63 were generated by the multi-criteria simplex method. All solutions are intended to be extreme nondominated points. However, due to rounding errors inherent is such methods, some dominated compromises may be generated. However, it always generates

[4]Analytically, one can determine that the librarian's duration received a weight of 94 percent, while the patron received a weight of six percent. Hence, an analytical utility function would choose the same resolution if these weights were associated with the librarian's and patron's duration preference.

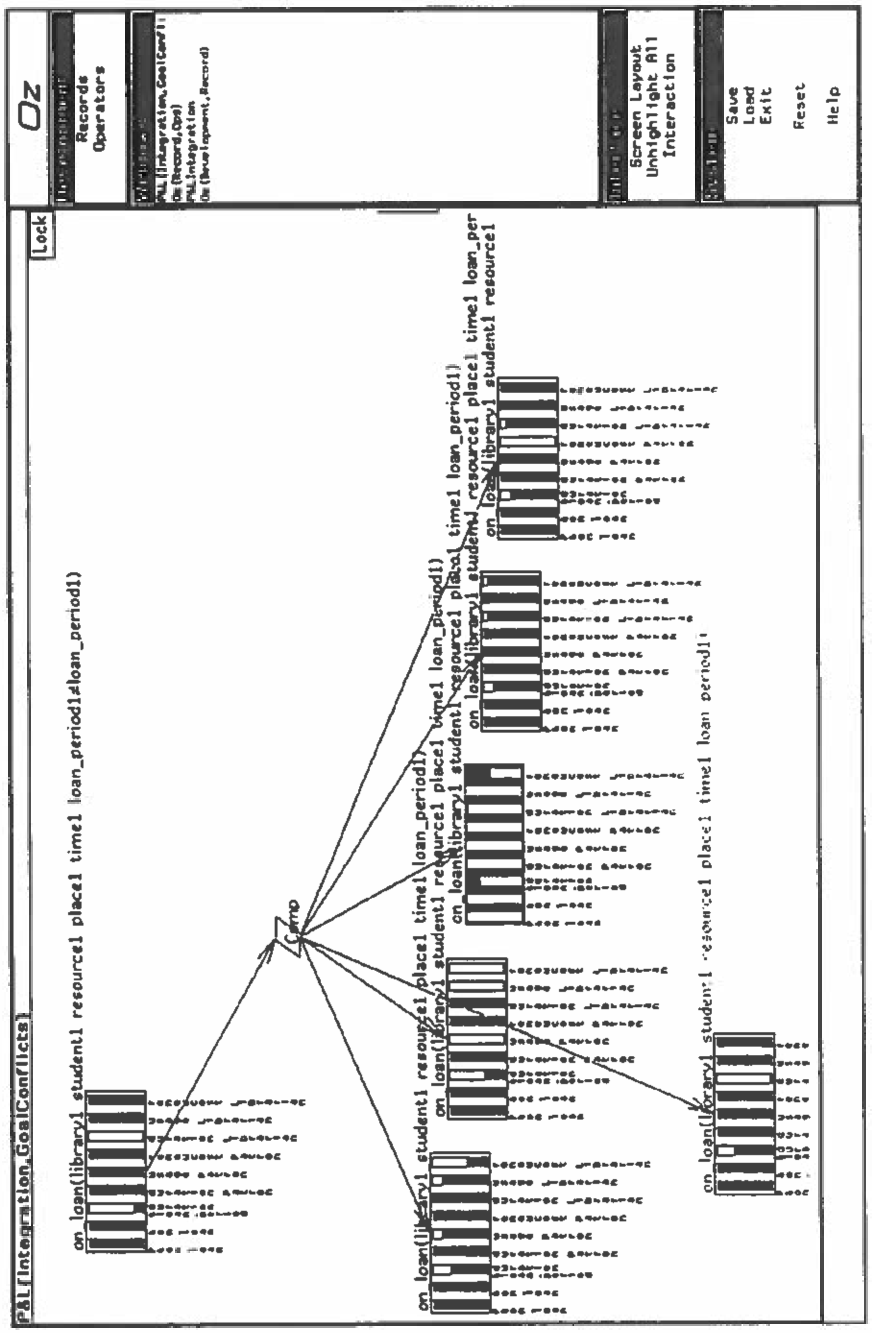

Oz
Records
Operators
P&L(Integration,GoalConfl
Oz(Record,Ops)
P&L(Integration)
Oz(Development,Record)
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help
P&L(Integration,GoalConflicts)
Lock
on loan(library1 student1 resource1 place1 time1 loan_period1≠loan_period1)
Comp
on loan(library1 student1 resource1 place1 time1 loan_period1)
on loan(library1 student1 resource1 place1 time1 loan_per

Figure 62. Oz Screen Depiction of the Compromised Loan Duration.

feasible resolutions. Further, it would be a simple matter to prune such dominated solutions by comparison with the goals.

After considering the various alternatives, the arbitrator can select a resolution. Instead, I assume that Burkhalter and Race continued on, leaving resolution selection until after both conflicts had their compromises generated.

2.1.4.2. Late Notice

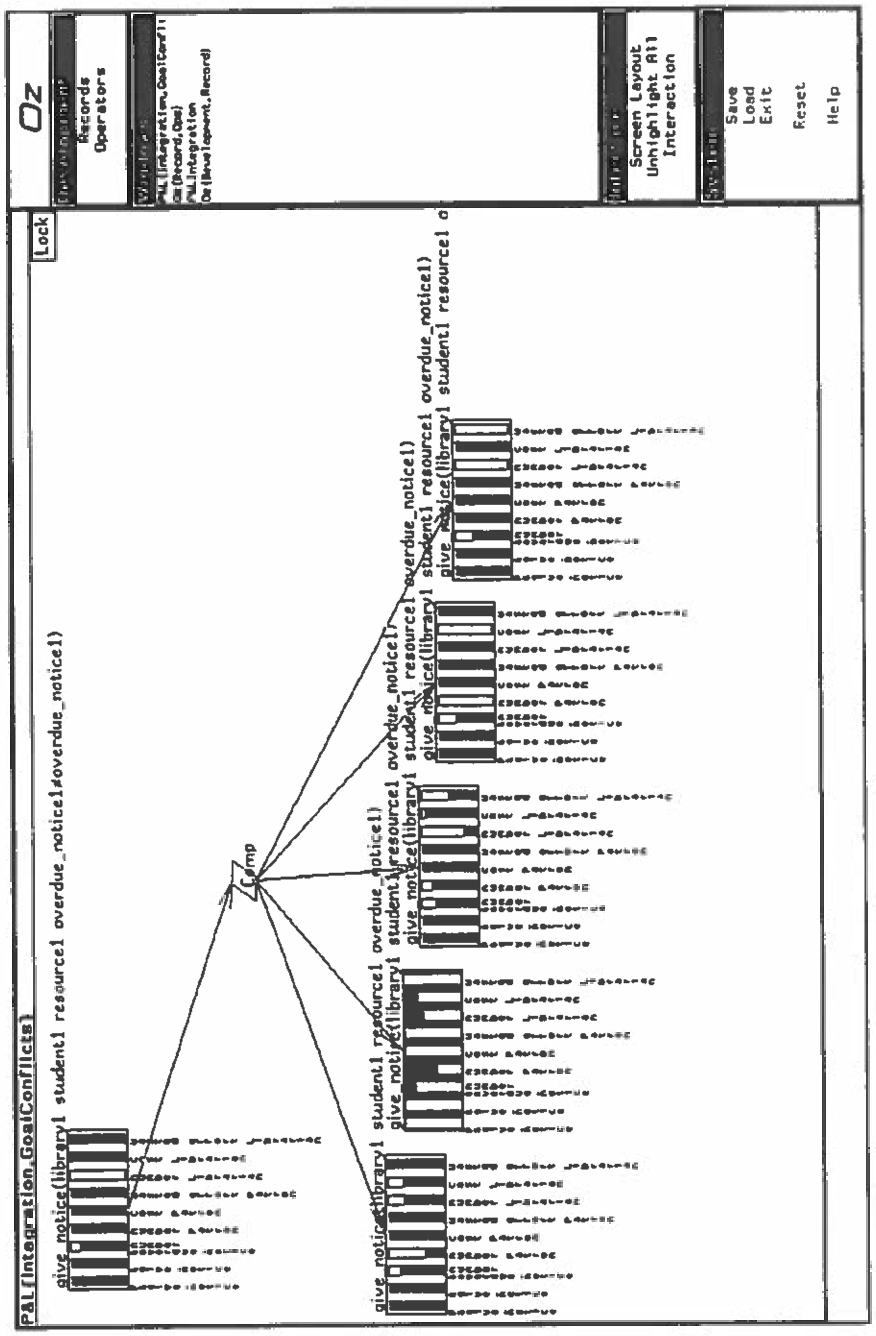
Figure 64 shows an Oz depiction of the overdue notice number compromises. Figure 65 shows the table of compromise resolutions depicted in figure 64. Again, these resolutions are generated by the multi-criteria simplex method. All solutions in the table are extreme nondominated points.

2.1.5. Resolution Choice

After considering the compromised resolutions for the two conflicts, Burkhalter and Race choose a resolution. They picked the 21 day loan period and the 2 overdue notices. The 21 day loan period (rounded) also happens to be an extreme point, so it is easily accessible from the set of displayed resolutions. In figure, 62 the arbitrator can simply choose the 21 day resolutions.

#	Duration	Usage	Renew Cost
1	365	100	0
2	43.75	96.73	0
3	34.60	96.64	10.46
4	20.97	96.50	45.71
5	14.03	80.11	63.69
6	0.0	0.0	100.0

Figure 63. Table of Loan Period Compromises.



Oz
Records
Operators
P&L[Integration,GoalConflicts]
Oz[Record,Goal]
P&L[Integration]
Oz[Development,Record]
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help
Lock
P&L[Integration,GoalConflicts]
give_notice(library1 student1 resource1 overdue_notice1&overdue_notice1)
Comp
give_notice(library1 student1 resource1 overdue_notice1)

Figure 64. Oz Screen Depiction the Compromised Notice Number.

#	Number	Cost	Wasted Effect
1	5.00	2.00	100.00
2	3.96	10.33	51.04
3	1.91	26.77	3.81
4	1.61	29.11	3.22
5	0.00	100.00	0.00

Figure 65. Table of Notice Number Compromises.

After doing, so it is highlighted (the image is inverse of the others).

Similarly, the overdue notice number of 1.91 is rounded to 2. Figure 63 illustrates the overdue resolution as highlighted. While the extremes 5 and 0 are shown, all infinite compromises between are not. Instead the arbitrator must do the rounding to 2 overdue notices. The arbitrator does so by selected the relation and setting its value. After doing so, the preference bars are updated. In this way, non-extreme nondominated compromises can be chosen.

2.1.6. Resolution Implementation

Once, the resolutions are chosen, they can be implemented. To do so, the arbitrator selects *create integrated model* from a menu, selects the the integration object in the development graph of figure 57, and hits the go button. First, a new perspective containing the resolved goals is automatically generated. Second, a design is automatically created satisfying those integrated goals.

The integrated model is a copy of one of the initial perspective's hierarchies, a (possibly transformed) initial state, and transformed goals. Since, it is assumed that all perspectives have the same initial conditions and hierarchies, the integrated perspective is the same as the original perspectives, but with the resolved goals. Only, hierarchy preferences go unnegotiated. However, since operator preferences are not a working part of the implementation, they will not effect the derived design. Hence, the integrated perspective represents the negotiated goals of the original perspectives and the generic library model.

The integrated goals are created by transforming each resolved relation based on the conflicting relation from which it was derived. In this example, the transformations are simple substitutions. This is because only the object attributes of the conflicting relations have changed. Hence, the two conflicting goals:

```
on_loan(student1,resource1,loan_period1.duration=14) ≠
on_loan(student1,resource1,loan_period1.duration=365)

give_notice(library1,student1,overdue_notice1.number=3) ≠
give_notice(library1,student1,overdue_notice1.number=5)
```

are transformed to the resolved goals of:

```
on_loan(student1,resource1,loan_period1.duration=21)

give_notice(library1,student1,overdue_notice1.number=2)
```

Once the integrated perspective is constructed, the planner can attempt to create a design achieving the integrated goals. When each goal interference is confined to corresponding goal conflicts (e.g., two-way interference) and the integrated goals are simple substitutions of the original goals, the planner is guaranteed to derived an integrated design. However, if two corresponding goals not only interference with each other, but with other conflicting goals (e.g., they all consume a single relation instance), then the planner may not be able to achieve an integrated design. Additionally, if the goals were transformed in ways other than simple substitution, as shown in the expanded example, then the planner may not be able to achieve an integrated design. However, in this simple example, neither of these cases hold, so the planner does derive a design. In fact, the integrated perspective and design look identical to that of figure 56.

2.1.7. Summary

In sum, in the strict rederivation:

(1) Oz detected two conflicts. Loan period duration and overdue notice number had different values in the librarian's and patron's designs.

(2) The arbitrator choose the compromise method. For each conflict, the arbitrator chooses to apply the compromise generation method.

(3) Oz generated compromises. Several compromise resolutions were generated for each conflict.

(4) The arbitrator choose two resolutions. Considering the satisfaction each perspective derived from the alternative resolutions, the arbitrator chooses one resolution for each conflict.

(5) Oz derived an integrated perspective and design. Given the choice of resolutions, Oz transformed the conflicting perspectives to create an integrated perspective, and then used the integrated perspective to derive an integrated design.

2.2. Expanded Derivation

In this expanded derivation, Oz again integrates the perspectives and designs of the librarian and patron. The first few events are identical to the previous derivation. First, goal correspondences are identified and their surface conflicts are noted. Next, plan interference between the goals is identified. Then, the conflicts are presented to the arbitrator. Next, compromises are generated.

After compromises are generated, this expanded derivation diverges from the strict derivation. While this derivation still picks the two overdue notice resolution, it continues the resolution process for loan period duration. Rather than simply choosing one of the compromises, the arbitrator applies the resolution specialization and generalization methods. While demonstrating these method, this expanded derivation also corresponds to: (1) the analysis of Burkhalter and Race, and (2) comments provided by an expert librarian. Specifically, the loan period for students will be specialized into two classes: undergraduates and graduates. Then, two different loan period durations will be given to each class. Finally, a new perspective and design are derived representing these expanded negotiations.

Rather than repeat the first few steps, this derivation will begin after the compromises are generated in section 2.1.4, and after the arbitrator has already chosen the 2 overdue notice resolution. Hence, the following derivation only considers the specialization and generalizations of the loan period duration relation.

2.2.1. Generated Specializations

The specialization method is based on the premise that a conflict over values associated with an abstraction can be resolved by assigning some of the conflicting values to some specializations of the abstraction. For the loan period conflict, specializations include subclasses of resource and student objects. One resolution, then, is to assign different loan period duration values to graduate and undergraduate students.

Figure 66 shows an Oz depiction of ON_LOAN specialized relations. These relations are based on specializations of the objects within the ON_LOAN relation. According to library model, the only objects in the ON_LOAN relation with specializations are student and resource. Student has

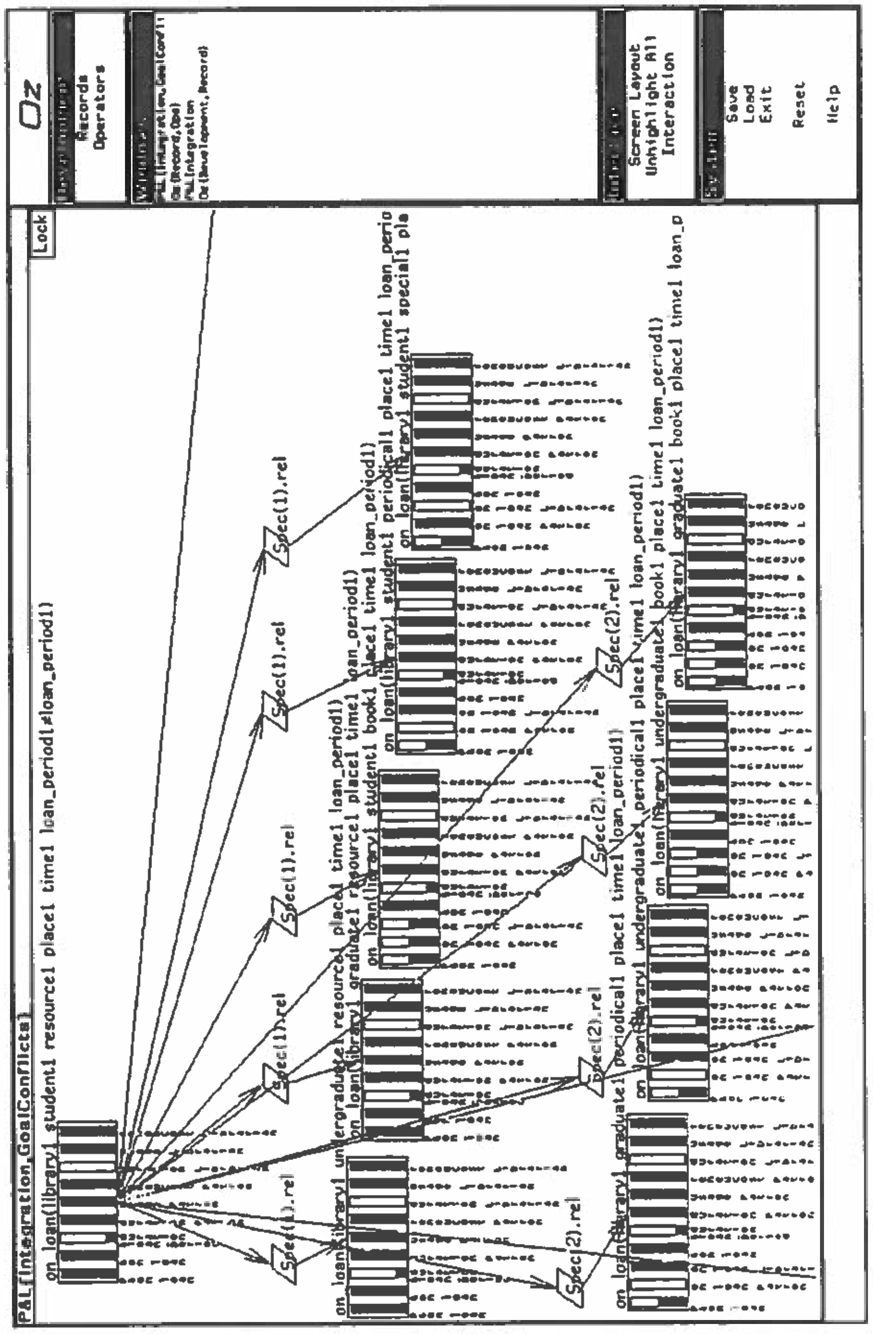

P&L[Integration,GoalConflicts]
Lock
on loan(library1 student1 resource1 place1 time1 loan_period1#loan_period1)
Spec(1).rel
Spec(2).rel
on loan(library1 undergraduate1 resource1 place1 time1 loan_period1)
on loan(library1 graduate1 resource1 place1 time1 loan_period1)
on loan(library1 student1 book1 place1 time1 loan_period1)
on loan(library1 student1 periodical1 place1 time1 loan_period1)
on loan(library1 graduate1 periodical1 place1 time1 loan_period1)
on loan(library1 undergraduate1 periodical1 place1 time1 loan_period1)
on loan(library1 undergraduate1 book1 place1 time1 loan_period1)
on loan(library1 graduate1 book1 place1 time1 loan_p
Oz
Records
Operators
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help

Figure 66. Oz Screen Depiction the of Specializations.

specializations undergraduate and graduate, while resource has specializations book, periodical, and special. Figure 66 illustrates the specialization of the conflicting relation by each of these objects along with the annotation `Spec(1)`. This indicates that these specialization are a distance of one specialization link away from the conflicting relation. Combinations of the two objects are two links away. To apply the specialization method, the arbitrator must supply the depth of specializations requested; the default is all specializations.

For the most part, the specialized relations have the same preferences as the original relation. For example, the loan period duration preference for undergraduates is the same in both perspectives as that for students. However, to illustrate how specialized preferences can appear during negotiations, we have attached opposing resource preferences to the ON_LOAN relation. One may notice from figure 66 that some ON_LOAN relation preferences have been specified. Those preferences are:

```
(in-perspective 'Librarian)

(def_mod_relation  on_loan(library patron resource place time loan_period)
   :order '(on_loan(library patron special place time loan_period)
            on_loan(library patron book place time loan_period)
            on_loan(library patron periodical place time loan_period))
   :objective 'max)


(in-perspective 'Patron)

(def_mod_relation  on_loan(library patron resource place time loan_period)
   :order '(on_loan(library patron special place time loan_period)
            on_loan(library patron book place time loan_period)
            on_loan(library patron periodical place time loan_period))
   :objective 'min)
```

They indicate that, given an exclusive choice of among the subclasses of resource, which resource subclasses are most preferred. For example, the patron prefers special resources over the other two, while the librarian prefers to loan periodicals over the other two. Given these preferences and a resource subclass conflict, one could automatically generate the compromise of books. Additionally, these preferences could have been used in an even more specialized resolution concerning trade-offs between librarian resource preferences and patron duration preferences. However, in this example, the arbitrator is not interested is such specialized resolutions. Instead, the focus will be on resource preferences for undergraduates and graduates.

Figure 67 presents a table of all the specializations derived from the ON_LOAN conflict. The arbitrator is most interested in continuing the exploration of resolutions along the lines of parameterizing loan period duration based on student subclasses. Hence, resolutions numbered one and two in figure 67 are the targets of the next resolution operator.

At this point, the arbitrator chooses to apply the compromise operator to both resolutions one and two. This results in the same compromises as shown in figure 63, but for student subclasses undergraduate and graduate. Once these resolutions are generated, the arbitrator chooses the 14 day duration for undergraduates and the 21 day duration for graduates. Figure 68 shows an **Oz** depiction of these resolutions. (The chosen resolutions are highlighted by inverting their

#	Level	Relation
1	1	on_loan(library1,undergraduate1,resource1,loan_period1)
2	1	on_loan(library1,graduate1,resource1,loan_period1)
3	1	on_loan(library1,student1,book1,loan_period1)
4	1	on_loan(library1,student1,periodical1,loan_period1)
5	1	on_loan(library1,student1,special1,loan_period1)
6	2	on_loan(library1,undergraduate1,book1,loan_period1)
7	2	on_loan(library1,undergraduate1,periodical1,loan_period1)
8	2	on_loan(library1,undergraduate1,special1,loan_period1)
9	2	on_loan(library1,graduate1,book1,loan_period1)
10	2	on_loan(library1,graduate1,periodical1,loan_period1)
11	2	on_loan(library1,graduate1,special1,loan_period1)

Figure 67. Table of On Loan Specializations.

images.)

2.2.2. Generated Generalizations

The generalization method is based on the premise that a conflict over values can be resolved by either: (1) abstracting the conflicting object, or (2) choosing one of the conflicting objects and giving something else to the "loser" of the negotiated conflict. For the loan period conflict, generalizations include parent relations of ON_LOAN relation, and relations asserted by operators similar (nearby) to the BORROW_RESOURCE operator. One resolution, then, is to suggest that agents recall resources that have been granted to other agents.

Figure 69 shows Oz depictions of ON_LOAN generalized relations. These relations are based on generalizing objects within the ON_LOAN relation. According to library model, the only objects in the ON_LOAN relation with generalizations are student and library. Student has generalizations patron and agent, while library has the generalization agent. Figure 69 illustrate the generalizations of the conflicting relation by each of these objects along with the annotation `Gen(1)`. This indicates that these relations are a distance of one generalization from the conflicting relation. Combinations of the two objects are two links away. To apply the generalization method, the arbitrator must supply the depth of generalizations requested; the default is all generalizations.

Figure 70 presents a table of all the generalizations derived from the ON_LOAN conflict. The first two resolutions are derived from the generalizations of the student and library. The remaining resolutions are derived through the operator hierarchy. In some cases, relations besides the key relation are shown. For example, the resolution suggesting that agents should renew their resources, generalization 4, is shown with the other relations that the renew operator asserts. Such resolutions are found by first finding similar operators to that which asserted the conflicting

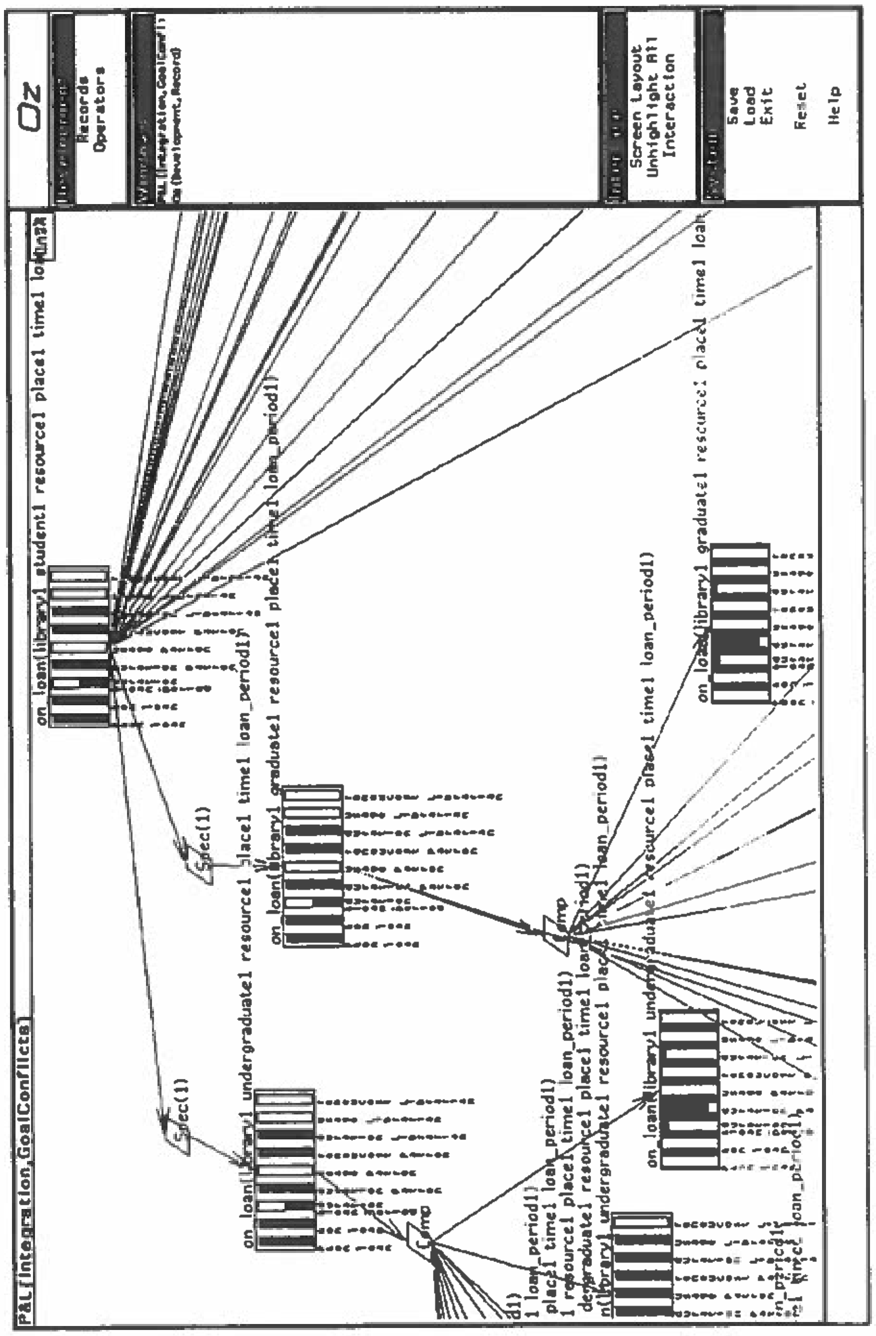


Figure 68. Oz Screen Depiction the Expanded Resolution Choice.

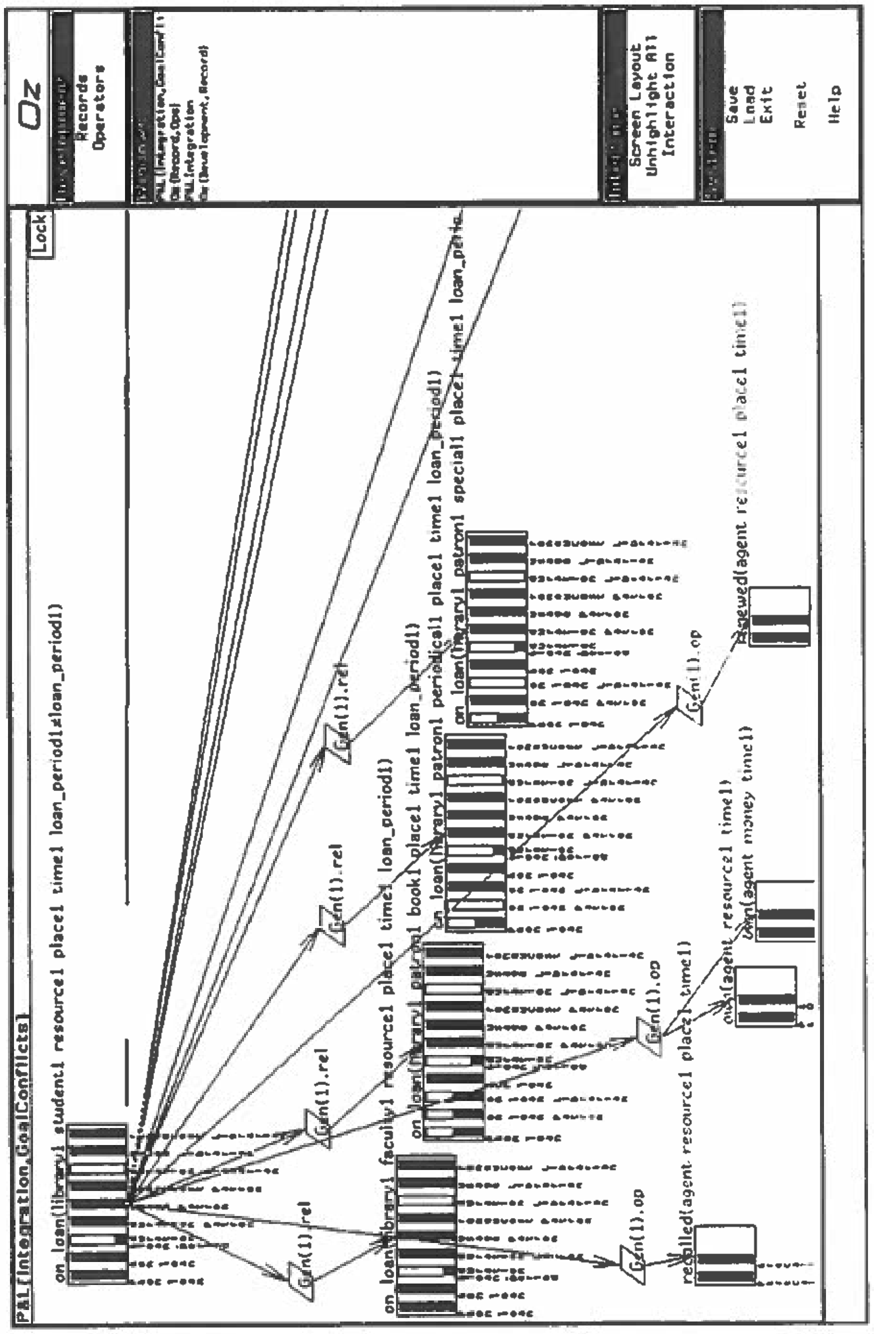

Oz
Records
Operators
Screen Layout
Unhighlight All
Interaction
Save
Load
Exit
Reset
Help
Lock
PAL[Integration,GoalConflicts]
on_loan(library1 student1 resource1 place1 time1 loan_period1&loan_period1)
on_loan(library1 faculty1 resource1 place1 time1 loan_period1)
on_loan(library1 patron1 book1 place1 time1 loan_period1)
on_loan(library1 patron1 periodical1 place1 time1 loan_period1)
on_loan(library1 patron1 special1 place1 time1 loan_perio
Gen(1).rel
Gen(1).op
recalled(agent resource1 place1 time1)
ask(agent resource1 time1)
own(agent money time1)
renewed(agent resource1 place1 time1)

Figure 69. Oz Screen Depiction the of Generalizations (1).

relation (i.e., BORROW_RESOURCE), and then presenting those asserted relations. In this way, alternative functionality is suggested as compensation to the conflicting goal.

2.2.3. Resolution Choice

At this point, the arbitrator determines that the generalized resolutions are not satisfactory. Instead, based on all the alternatives considered thus far, the arbitrator stops the resolution process by choosing the specializations of the previous section (figure 67).

2.2.4. Resolution Implementation

Like in the first derivation, once the resolutions are chosen, they can be implemented. The integrated model is a copy of one of the initial perspective's hierarchies, a transformed initial state, and transformed goals. The integrated goals are created by transforming each resolved relation based on the conflicting relation from which it was derived. In this example, the transformations are object substitutions. This is because the initial conflicting goals have been specialized. The two conflicting goals:

#	Level	Relation
1	1	on_loan(library1,patron1,resource1,loan_period1)
2	1	on_loan(agent1,patron1,resource1,loan_period1)
3	1	on_loan(library1,patron1,book1,loan_period1)
4	1	on_loan(library1,patron1,periodical1,loan_period1)
5	1	on_loan(library1,patron1,special1,loan_period1)
6	1	on_loan(library1,faculty1,resource1,loan_period1)
2	1	own(agent1,resource1)
3	1	recalled(agent1,resource1)
4	1	renewed(agent1,resource1)

Figure 70. Table of On Loan Generalizations.

```
on_loan(student1,resource1,loan_period1.duration=14) ≠
on_loan(student1,resource1,loan_period1.duration=365)

give_notice(library1,student1,overdue_notice1.number=3) ≠
give_notice(library1,student1,overdue_notice1.number=5)
```

are transformed to the resolved goals of:

```
on_loan(undergraduate1,resource1,loan_period1.duration=21)
on_loan(graduate1,resource1,loan_period1.duration=14)

give_notice(library1,undergraduate1,overdue_notice1.number=2)
give_notice(library1,graduate1,overdue_notice1.number=2)
```

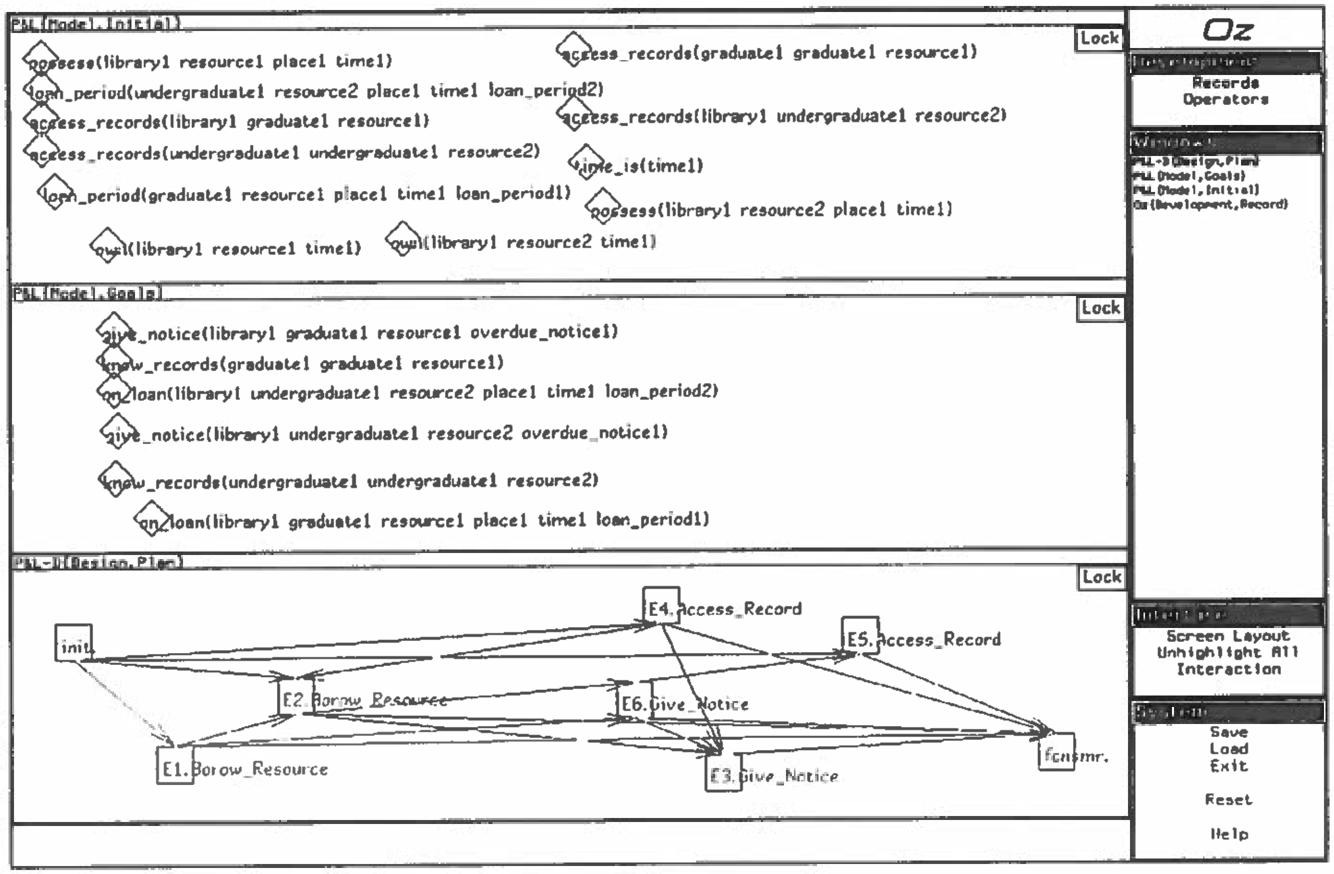
Additionally, the initial library state is transformed. For each reference to student1, two new relation instances are created which refer to undergraduate1 and graduate1. When the transforming relations to use undergraduate rather than student, any references to loan period duration will also be set to 14. Similarly, graduate relations will have their duration's set to 21.

Once the integrated perspective is constructed, the planner can attempt to create a design achieving the integrated goals. Figure 71 shows an Oz depiction of the integrated perspective including the initial state. From this description the planner can derive the design also shown in figure 71. Notice the use of a different BORROW_RESOURCE operator for each of the two student types. This is because of the planner's operator representation. For each different relation, an single operator is used; hence, the different student loan periods are asserted by different operators.

2.2.5. Summary

In sum, in the expanded derivation:

(1) Oz detected two conflicts. Loan period duration and overdue notice number had different values in the librarian's and patron's designs.



PAL(Model,Initial)

possess(library1 resource1 place1 time1)
loan_period(undergraduate1 resource2 place1 time1 loan_period2)
access_records(library1 graduate1 resource1)
access_records(undergraduate1 undergraduate1 resource2)
loan_period(graduate1 resource1 place1 time1 loan_period1)
own(library1 resource1 time1)
access_records(graduate1 graduate1 resource1)
access_records(library1 undergraduate1 resource2)
time_is(time1)
possess(library1 resource2 place1 time1)
own(library1 resource2 time1)

PAL(Model,Goals)

give_notice(library1 graduate1 resource1 overdue_notice1)
know_records(graduate1 graduate1 resource1)
on_loan(library1 undergraduate1 resource2 place1 time1 loan_period2)
give_notice(library1 undergraduate1 resource2 overdue_notice1)
know_records(undergraduate1 undergraduate1 resource2)
on_loan(library1 graduate1 resource1 place1 time1 loan_period1)

PAL-D(Design,Plan)

Figure 71. Oz Screen Depiction the Expanded Integrated Model.

(2) The arbitrator choose the compromise method. For each conflict, the arbitrator choice to apply the compromise generation method.

(3) Oz generated compromises. Several compromise resolutions were generated for each conflict.

(4) The arbitrator choose the specialization method.

(5) Oz generated specializations.

(6) The arbitrator focused on two resolutions.

(7) The arbitrator choose the compromise method. For each resolution, the arbitrator choice to apply the compromise generation method.

(8) Oz generated compromises. Several compromise resolutions were generated for each resolution.

(9) The arbitrator choose the generalization method.

(10) Oz generated generalizations.

(11) The arbitrator choose three resolutions. Considering the satisfaction each perspective derived from the alternative resolutions, the arbitrator choose the two loan period resolutions and the one overdue notice resolution.

(12) Oz derived an integrated perspective and design. Given the chosen of resolutions, Oz transformed the conflicting perspectives to create an integrated perspective, and then used the integrated perspective to derive an integrated design.

2.3. Comparative Evaluation

In evaluating the above derivations, I consider four aspects:

(1) Conflict Perception Comparison

Conflict detection by Oz was identical to that of the actual Burkhalter and Race study. The loan period and overdue notice convicts were detected while the rest of the library was ignored. However, this assumes that the rest of the polices in the original case study were non-conflicting. Not doubt, this is not true. It is more likely that some polices were conflicting, but Burkhalter and Race simply ignored them and focused on what they felt were relevant issues. Moreover, it appears that Burkhalter and Race did not initially detect the overdue notice conflict. Instead, analysis of the relationship between renewal and loan period duration led them to overdue notice number. From this I conclude that Oz is more likely to detect all conflicts, but not be able to determine which can be ignored and which should be resolved. Note that such detection does not require that Oz have the cost effectiveness relationships of section one. Only the goal relationships, indirectly determined through the planner, are necessary for conflict detection. Hence, while Oz appears to be an effective functional conflict detector, it does not have knowledge to determined which are relevant. This is typical of the philosophy of its decision support. Oz presents information, but does not attempt to make decisions. However, Oz does allow the arbitrator to ignore conflicts. To do so, an arbitrator can simply choose one perspective's values without engaging the negotiation methods.

(2) Resolution Generation Comparison

Compromise generation by Oz was identical to that of the actual Burkhalter and Race study. This was mainly due to the encoding of the analysis made in the original study. (Oz has no means to derive the various cost and effectiveness graphs from a initial designs.) However, the expanded example suggested resolutions not found in the original study. While Burkhalter and Race did conduct analysis of resource usage for undergraduate and graduate subclasses, they combined the analysis when they derived their resolution. No reason was given for this; however, it is likely that the similarity of the specialized graphs led them to coalesce there analysis. The additional resolutions suggested may also have been consider by Burkhalter and Race; however, it is likely that they recognized that most of the other resolutions were currently in the library (e.g., recall, renewal) or should not be part of an academic library (e.g., selling resources).

(3) Resolution Choice Comparison

Resolution choice in the strict derivation was identical to that of the actual Burkhalter and Race study. This was not a parameterized part of the automated study. The resolution was chosen to simulate the original study. The expanded derivation included the parameterized student loan periods. This choice was made to simulate the University of Oregon's library policy.

(4) Rationale Record Comparison

The rationale record of Oz is obviously much richer than that of Burkhalter and Race. While their study has provided an exception description of their redesign (one of the reasons for choosing it), they did leave much out. In contrast, Oz provides records of: (1) the initial perspectives, designs, and conflicts, (2) the alternatives considered, (2) the order they were considered, (3) the resolutions chosen, and (4) the new design.

Based on this comparative analysis, I conclude that the representations and algorithms do indeed support both the negotiation design process, but also records of its occurrence.

2.4. A Librarian's Observations

To confirm my intuition about the case study, I consulted a librarian. This consultation consisted of: (1) presentation of the Burkhalter and Race case study, and (2) presentation of the rederivation. The librarian was not informed of the expanded derivation. For the librarian's part, she considered the original study and its rederivation, but did not directly analyze the rederivation step by step. Generally, the librarian agreed with above conclusions. Additionally, she made the following observations:

(1) The quantitative analysis conducted by Burkhalter and Race is generally not done. It is time consuming and error prone. However, when such studies are available, librarians use them to: (1) make choices, and (2) rationalize their choice to others. Hence, if Oz could generate some of the analysis, even qualitatively, it would be of great assistance.

(2) Conflict detection is of lesser concern. Librarian's believe they understand how policies conflict through the feedback from their patrons.[5] However, the resolution procedure was of greater interest. Specifically, the generation of analytic compromises seemed appealing. Not so much as a decision aid, but as a means of convincing others that their policies were correct.

(3) Qualitative conflict generation appeared more appealing. In considering the case study, the librarian suggested that Burkhalter and Race should have considered dividing students into to subclasses, particularly, graduates and undergraduates. She suggested this

[5]This may or may not be true. As noted previously, Burkhalter and Race did not initially realize the ineffectiveness of the third overdue notice. Only after the loan period conflict was analyzed, did overdue notice number become an issue.

alternative without prompting, perhaps, partly because the University of Oregon's library has such a policy. Additionally, she was slightly surprised that Burkhalter and Race did implement the three week loan period. (In fact, the University of Oregon's six month faculty and graduate student loan policy is currently under consideration for reduction.) One reason she cited against longer loan periods was that patrons can simply renew their resources (once) to obtain twice the duration. However, this was just the process that Burkhalter and Race were trying to reduce. While it is true that their study was some years before library automation (1968), it is still the true today that the renewal process uses circulation staff resources. Any reduction in renewals is bound to free up some circulation resources.

In sum, the librarian found the (experimental) system appealing. She suggested that such a system might readily fit into current efforts aimed at automating a library. For example, she suggested that some of the quantitative analysis conducted by Burkhalter and Race could be retrieved directly using existing library software. In the original case study, the circulation records and surveys were used to determine the relationships between loan period, renewal, and resource usage. Some of these relationships can be inferred from the recorded circulation data (e.g., reduced loan periods increases renewals, and increased renewals increases reduces circulation staff resources.)

3. Summary

The evaluation of the two derivations suggest that the representations and algorithms presented in this dissertation can provide automated assistance for negotiated design. Specifically, the algorithms automate:

(1) Conflict detection.

(2) Resolution generation.

(3) Resolution implementation.

(4) Negotiation rationale.

CHAPTER VII

CONCLUSIONS

This dissertation presented a methodology and automated algorithms for collaborative design. The methodology called for individuals to independently create designs achieving their own goals, and then collectively derive a single unified design using automated negotiation techniques. From a software engineering perspective, the methodology provides parallelism, simplicity, rationale, and reuse. From a negotiation perspective, the methodology provides multiple agent preference maximization and novel resolution synthesis. From an artificial intelligence perspective, the algorithms provide automation for the complex processes of conflict detection, resolution synthesis, and resolution selection. This dissertation described how interests of individuals or subgroups can productively aid the derivation of robust collaborative designs through the automated negotiation of their conflicts.

This dissertation described formal representations for modeling individual perspectives, design conflicts, and subtasks involved in negotiation. Specifically described were representations for: (1) goals and preferences over domain operators, objects, and relations, (2) categories of design and goal conflicts, and (3) categories of conflict resolutions. Automated processes can manipulate these representations to aid group negotiation.

This dissertation described formal algorithms for detecting conflicts and synthesizing resolutions. Specifically described were algorithms for: (1) distinguishing between simple design differences and design interference, (2) mapping between goals and their supporting design components, (3) detecting goal conflicts, (4) synthesizing analytic and heuristic resolutions, and (5) reintegrating resolved goals into a design. Analytic resolution consists of compromise generation using a multiple criteria linear programming method. Heuristic resolution consists of search through domain hierarchies to synthesize dissolutions and compensations. These methods have been implemented and applied.

This dissertation describes the implementation of negotiation algorithms and their application to library design problems. The design of library systems is a complex, multiple agent, negotiation enterprise. I have represented portions of documented library designs in the collaborative design tool, Oz. Oz has been used to detect conflicts and derive negotiated resolutions similar to those published by expert librarians. The implementation and its application to the library domain support the central tenet of this dissertation: processes of negotiated design can be automated through the representation of a generic domain model and specific representations of individual perspectives.

1. Contributions

Research presented in this dissertation has contributed new ideas concerning design methodology and automated negotiation. It argues for independent design followed by negotiated integration. It describes how automated support can be provided for this methodology and presents a case-study as demonstration. Most importantly, this dissertation illustrates how, heretofore implicit, negotiation processes can be made explicit and supported.

Oz is a product of this dissertation. It demonstrates how negotiated design can be supported through computer automation. Oz has successfully combined the following subsystems to form a negotiation design tool.

(1) Perspective modeling.

(2) Automated design derivation.

(3) Design integration using conflict detection, resolution generation, and resolution implementation.

Oz has been successful because of the following:

• Hierarchical Generic Domain Model

The use of domain hierarchies has: (1) simplified preference description, and (2) provided a basis for resolution search by interrelating domains so conflicts can be transformed meaningfully. Most importantly, the automatically constructed operator hierarchy provides a way to find functionally similar, yet syntactically different, goals.

• Interactive Resolution Procedure

Allowing the user to control the search for resolutions has unburdened the system from codifying, complex and abundant, domain and contextual dependent, knowledge concerning individual and group preference trade-offs. Instead, Oz generates possible resolutions while the user guides the search and chooses resolutions. This has proved to be a key decision, both for tool acceptance, and making the problem tractable.

• Simplifying Assumptions

The simplifying assumptions concerning common representation languages, simplified design language, and consistent terminology have eased the implementation task. However, they have not crippled the basic ideas. The negotiated design algorithms will scale up even where these assumptions do not hold as long as the input to the algorithms obey these assumptions. Moreover, the algorithms can be extended to multiple, more expressive, languages by describing how conflicts are defined in the languages and how goals are linked to designs. The basic ideas of conflict detection, characterization, and resolution do not depend on these assumptions.

2. Limitations

Some of the reasons that Oz has been successful are also limitations. This is mainly due to the limited resources of a dissertation. However, others are more fundamental problems.

• Simplified Languages

Problems of expressibility can be overcome by using more expressive perspective and design languages. For perspectives, non-linear and conditional constraints will help. Also, initial preference trade-offs should be recorded when they exist. For designs, incorporating time and recursive processes will help. In any case, Oz must be updated to: (1) detect meaningful goal differences, (2) understand the derivation of design components from goals, (3) detect meaningful design language interference. Once these can be defined for perspective modeling and design language, the Oz approach can be applied.

• Knowledge Engineering Bottleneck

Oz relies on the domain model to generate heuristic resolutions. Only the resolutions that can be synthesized by combining domain model components can be generated. Hence, the system is limited by the knowledge initially placed into it. Unfortunately, constructing the domain model can be a difficult and time consuming process. However, Oz may assist in this process, as suggested in chapter I, by posing hypothetical components which, if acquired, could resolve a conflict.

• Resolution Explosion

Oz has no way of controlling the resolution search process. While the user currently controls this process by choosing conflicts to work on and setting the resolution generation level, the user can still be overcome by the number of resolutions generated. To control heuristic search, Oz must use the individual preferences represented, infer or enquire about group preferences, and incorporate domain control heuristics.

3. Future Research

Future research entails expanding rationale capture, expanding automation, applying the methods to new domains.

• Expanding Rationale Capture

Since Oz is based on an interactive approach, some *a prior* knowledge is not captured. Currently, Oz only tracts initial goals and preferences through negotiations, but does not capture initial preference trade-offs. Individual's may wish to state their initial preference trade-offs, or even, define group trade-offs. Such a utility based approach is not supported, but would be useful even with the interactive approach. One could state initial trade-offs and tract them as they changed during design and group integration. In doing so, one could observe how the design and integration processes effected individual perspectives and how group trade-offs were derived from them. Additionally, utility analysis could be used to resolve unimportant conflicts, while a human directed the resolution of more important conflicts.

• Expanding Automation: Resolution Search Control

Currently, Oz is weak in two areas of automation: controlling resolution search and constructing domain models. During heuristic resolution generation, alternative relations are sought in the domain hierarchies. Constraining the resolutions considered, especially with an interactive approach, is essential. One approach is to use preferences attached to the relations to limit consideration: those relations which are dominated by more preferred relations can be thrown out. Domain dependent heuristics can also be used, e.g., resource selling should not be part of an academic library. Finally, distance measures and trade-off analysis can also be used: the user can specify that relations which reduce achievement of some criteria not be presented, or relations greater than some distance from the current (or ideal) alternative should not be presented.

• Expanding Automation: Domain Model Development

Domain model construction is also a problem. Currently, there is no aid to detect or resolve inconsistent preferences among the operator, object, and relation hierarchies. Attribute relationships must be manually defined (e.g., reduced loan period duration increases renewal costs). However, some of these relationships may be determined through simulation of plans involving such operators. Finally, the resolution process can require the description of previously undefined preferences or suggest the inclusion of new operators, objects, and relations. The role of the resolution process in modifying the domain model should be explored.

• New Domains

While **Oz** has only been applied to the library domain, its underlying domain model has been shown to capture knowledge from other domains[2]. However, it should be applied to other contexts besides design. For example, decision support for librarians or group scheduling. **Oz** can be applied to any domain where: (1) a domain model can be constructed, (2) goal conflict can be detected, (3) linkage between goals and designs can be established, and (4) design interference can be determined. Library decision support appears promising on these grounds. Moreover, some relationships of the domain model may be inferred from running library software systems. Librarians appear receptive to a negotiation aid, especially since their policies are continually be reconsidered and renegotiated. For similar reasons, group meeting scheduling appears promising. Individuals can categorize scheduled events into various operator types and describe criteria for good schedules (e.g., no time conflicts, no back-to-back meetings). However, in group scheduling, individual's meeting goals must be negotiated. For simple conflicts, an expanded Oz could run automatically, whereas, group resolution of more important conflicts could be interactively assisted.

4. Conclusion

This dissertation has presented a methodology for independent design and algorithms for its automated support. It has demonstrated that an implemented system can indeed provide users with support sufficient to generate previously negotiated designs. Its basic approach of knowledge-based interactive resolution shows promise. Both in design support, but also in other domains of group negotiation.

APPENDIX A

THE LIBRARY MODEL

```
(open_new_kb "Library")

(def_otyp "resource"
    :sb '("book" "periodical" "special"))

(def_otyp "thing"
 :sb '("money" "resource"))

(def_otyp "time"
    :d "Actual time, some day count from 0."
    )

(def_otyp "place")

(def_mod_otyp "overdue_notice"
 :at '(("Number" :val 3 :min 0 :max 5)
       ("Cost" :val 0 :min 0 :max 100)
       ("Wasted Effort" :val 0 :min 0 :max 100))
 :att_constraints '(
                  ;; Cost
                  (44 1 0 >= 100)
                  (8 1 0 >= 42)
                  ;; Wasted Effort
                  (-2 0 1 >= 0)
                  (-23 0 1 >= -40)
                  (-47 0 1 >= -135)))

(def_mod_otyp "loan_period"
 :at '(;;Ranges should be same for all models!
       ("Duration" :val 14 :min 0 :max 365)
       ("Usage" :val 100 :min 0 :max 100)
       ("RenewCost" :val 0 :min 0 :max 100))
 ;; These columns must be in the order of the above variables.
 :att_constraints '(
                  ;; Usage
                  (-5.71 1 0 <= 0)
                  (-2.36 1 0 <= 47)
                  (-0.01 1 0 <= 96.29)
                  ;; RenewCost
                  (2.5882 0 1 >= 100)
                  (1.1429 0 1 >= 50)))

(def_otyp "student"
    :d "Student patron types."
    :sb '("undergraduate" "graduate"))

(def_otyp "patron"
    :d "Patron types."
    :sb '("student" "faculty"))

(def_otyp "agent"
    :d "Active system agents."
    :sb '("patron" "library"))


(def_ptyp "time_is"
    :d "The time is some_integer."
    :o '("time"))

(def_ptyp_all "own"
    :o '("patron" "resource" "time"))

;;; Simplified own.
(def_ptyp "own"
    :o '("agent" "thing" "time")
    :sb '(("own"(library resource time))
         ("own"(patron resource time))))

(def_ptyp "renewed"
 :o '("agent" "resource" "place" "time"))

(def_ptyp "recalled"
 :o '("agent" "resource" "place" "time"))

(def_ptyp_all "loan_period"
    :d "A patron can borrow resource for a loan period DURATION"
    :o '("patron" "resource" "place" "time" "loan_period"))

;(def_ptyp_all "possess"
;    :d "Agent physcially has resource."
;    :o '("agent" "resource" "place" "time" "loan_period"))

;;; Simplify possess relation.
(def_ptyp "possess"
    :d "Agent physcially has resource."
    :o '("agent" "resource" "place" "time")
    :sb '(("possess"(library resource place time))
         ("possess"(patron resource place time))))

(def_ptyp_all "on_loan"
 :d "Resource on loan from TIME until TIME+ LOAN_PERIOD."
 :o '("library" "patron" "resource" "place" "time" "loan_period"))

;;; Simplify give_notice relation.
;(def_ptyp_all "give_notice"
; :d "LIBRARIANs give PATRONs a NOTICE."
; :o '("library" "patron" "resource" "overdue_notice"))

(def_ptyp_all "give_notice"
 :d "LIBRARIANs give PATRONs a NOTICE."
 :o '("library" "student" "resource" "overdue_notice"))

(def_ptyp "give_notice"
 :d "LIBRARIANs give PATRONs a NOTICE."
 :o '("library" "patron" "resource" "overdue_notice")
 :sb '(("give_notice"(library student resource overdue_notice))
       ("give_notice"(library faculty resource overdue_notice))))

(def_ptyp "read"
 :d ""
 :o '("agent" "resource" "place" "time" "loan_period"))

(def_ptyp_all "access_records"
 :d ""
 :o '("agent" "patron" "resource"))

(def_ptyp_all "know_records"
 :d ""
 :o '("agent" "patron" "resource"))

(def_ptyp "secure_records"
 :d ""
 :o '("agent" "agent" "resource"))


;;; These operators describe simply how two agents can carry out transactions
;;; for buying, borrowing, renewing, and recalling resources.

;;; Future developments include: subtypes of these operators, e.g. recall by
;;; phone or desk.
```

```
(def_etyp "Read_Resource"
    :d "An AGENT can read a RESOURCE at PLACE from a PERIOD from TIME."
    :o '(agent1 resource1 time1 place1 loan_period1)
    := '((possess(agent1 resource1 place1 time1)))
    :- '()
    :+ '((read(agent1 resource1 place1 time1 loan_period1))
        ))


(def_etyp "Recall_Resource"
    :d "A loaned RESOURCE is given up by an AGENT."
    :o '(agent1 agent2 agent3 resource1 time1 time2 place1 place2
              loan_period1 loan_period2)
    := '((time_is(time2))
        (loan_period(agent3 resource1 place1 time1 loan_period1)))
    :- '((possess(agent1 resource1 place1 time1))
        (on_loan(agent2 agent1 resource1 place1 time2 loan_period2)))
    :+ '((possess(agent3 resource1 place1 time2))
        (on_loan(agent2 agent3 resource1 place1 time2 loan_period1))
        (recalled(agent3 resource1 place1 time2))
        ))


(def_etyp "Renew_Resource"
    :d "An AGENT is given a new due date for a loaned RESOURCE."
    :o '(agent1 agent2 resource1 place1 loan_period1 time1 time2)
    := '((loan_period(agent1 resource1 place1 time1 loan_period1))
        (time_is(time2)))
    :- '((on_loan(agent2 agent1 resource1 place1 time1 loan_period1))
        (possess(agent1 resource1 place1 time1)))
    :+ '((on_loan(agent2 agent1 resource1 place1 time2 loan_period1))
        (possess(agent1 resource1 place1 time2))
        (renewed(agent1 resource1 place1 time2))
        ))

(def_etyp "Borow_Resource"
    :d "A lonable RESOURCE is LOANED to an AGENT for LOAN_PERIOD."
    :o '(agent1 agent2 resource1 time1 place1 loan_period1)
    := '((loan_period(agent1 resource1 place1 time1 loan_period1)))
    :- '((possess(agent2 resource1 place1 time1)))
    :+ '((possess(agent1 resource1 place1 time1))
        (on_loan(agent2 agent1 resource1 place1 time1 loan_period1))
        ))

(def_etyp "Buy_Resource"
    :d "Agents exchanges MONEY for RESOURCE."
    :o '(time1 time2 time3 agent1 agent2 resource1 money1
             place1 loan_period1 loan_period2)
    := '((time_is(time3)))
    :- '((own(agent1 resource1 time1))
         (possess(agent1 resource1 place1 time1))
         (own(agent2 money1 time2))
         (possess(agent2 money1 place1 time2)))
    :+ '((possess(agent2 resource1 place1 time3))
         (own(agent2 resource1 time3))
         (own(agent1 money1 time3))
         (possess(agent1 money1 place1 time3))
         ))


(def_etyp "Access_Record"
    :d "An AGENT can read the library loan records."
    :o '(agent1 agent2 resource1)
    :~ '((secure_records(agent1 agent2 resource1)))
    := '((access_records(agent1 agent2 resource1)))
    :- '()
    :+ '((know_records(agent1 agent2 resource1))
         ))

(def_etyp "Secure_Other_Records"
    :d "A PATRON can not read the library loan records."
    :o '(patron1 patron2 resource1)
    :~ '((know_records(patron1 patron2 resource1)))
    := '((access_records(patron1 patron2 resource1)))
    :- '()
    :+ '((secure_records(patron1 patron2 resource1))
         ))

(def_etyp "Secure_Self_Records"
    :d "A PATRON can not read their own library loan records."
    :o '(patron1 resource1)
    :~ '((know_records(patron1 patron1 resource1)))
    := '((access_records(patron1 patron1 resource1)))
    :- '()
    :+ '((secure_records(patron1 patron1 resource1))
         ))

(def_etyp "Give_Notice"
    :d "The LIBRARY gives PATRONs OVERDUE NOTICEs for RESOURCEs."
    :o '(library1 patron1 resource1 place1 time1 loan_period1 overdue_notice1)
    := '((on_loan(library1 patron1 resource1 place1 time1 loan_period1)))
    :- '()
    :+ '((give_notice(library1 patron1 resource1 overdue_notice1))
        ))


(def_oz_prob 'loan-and-records
 :o '((library1 library)
      (student1 student)
      (resource1 resource)
      (resource1 resource)
      (time1 time)
      (place1 place)
      (overdue_notice1 overdue_notice)
      (loan_period1 loan_period))
 :i '((possess(library1 resource1 place1 time1))
      (loan_period(student1 resource1 place1 time1 loan_period1))
      (own(library1 resource1 time1))
      (access_records(library1 student1 resource1))
      (access_records(student1 student1 resource1))
      (time_is(time1)))
 :g'(
     (give_notice(library1 student1 resource1 overdue_notice1))
     (know_records(student1 student1 resource1))
     (on_loan(library1 student1 resource1 place1 time1 loan_period1))
     )
 )

(set_default_strategy)
```

APPENDIX B

A DETAILED MCSM EXAMPLE

While the loan period examples illustrates the interface between design space and compromise space, the following example focuses on compromise search. Below, is a production mix example; two widgets x_1 and x_2 are to be produced subject to three resouce constraints. Two objectives o_1, and o_2 are to be maximized, and one, o_3, is to be minimized.

Example:

```
(def_order x1 VALUES (0 ... 2500) #'identity_order)
(def_scale x1 VALUES (x1 VALUES) #'identity_scale)
(def_order x2 VALUES (0 ... 2500) #'identity_order)
(def_scale x2 VALUES (x2 VALUES) #'identity_scale)

(def_constraint c1 (10 x1 + 6 x2 ≤ 2500))
(def_constraint c2 (5 x1 + 10 x2 ≤ 2500))
(def_constraint c3 (7 x1 + 7 x2 ≤ 2050))

(def_establish o1 max (10 x1 + 20 x2))
(def_establish o2 max (23 x1 + 32 x2))
(def_establish o3 min (x1 + x2))
```

The example consists of scales x_1 and x_2, constraints c_1, c_2, and c_3, and objectives o_1 and o_2.

Even with our scale, constraint, and objective representation, some converion is required to begin the Multi-criteria Simplex Method (MSM). First, inequalities expressed in constraints must be expressed as equalities by appending slack variables. For example, $10x_1 + 20x_2 \leq 30$ would be transformed to $10x_1 + 20x_2 + x_3 = 30$, where x_3 is a slack variable. Slack variables are distinct from decision variables which occur in the original problem description. A multi-criteria simplex tableau is constructed using the coefficients and values found from the problem criteria and constraints. Variable coefficients are divided into basic and nonbasic groups. In an initial MSM tableau, nonbasic variables will represent decision variables, basic variables will represent slack variables. Basic variables form the current "basis" of a solution, their values will be represented in the last column of the tableau. The last row represents the objectives; in the case of maximization, their coeficents are negated.

Below, the general MSM tableau is illustrated. Basic variables are named x_1 ... x_m, nonbasic variables x_{m+1} ... x_n, cooefficients of the nonbasic constraints y_{ij}, coefficients of the nonbasic objectives z_{ij}. Values are named $x_m{}^k$ where *m* is the refers to the variables and *k* refers the the *k*th tableau, or solution set.

Current Basis	Basic variables $x_1 \ldots x_m$	Nonbasic variables $x_{m+1} \ldots x_j \ldots x_n$	Values of basic variables
x_1	$1 \ldots 0$	$y_{1(m+1)}$ $y_{1j} \cdots y_{1n}$	x_1^0
x_m	$0 \ldots 1$	$y_{m(m+1)}$ $y_{mj} \cdots y_{mn}$	x_m^0
Criteria Rows	$1 \ldots 0$	$z_{1(m+1)}$ $z_{1j} \cdots z_{1n}$	$f_1(x^0)$
	. .	. . .	.
	. . .	. . .	.
	$0 \ldots 0$	$y_{l(m+1)}$ $y_{lj} \cdots z_{ln}$	$f_l(x^0)$

Our problem can be formulated as the following multi-objective linear programing problem:

$$\begin{array}{lrcrcr}
\text{Max} & 10\,x_1 & + & 20\,x_2 & & \\
\text{Max} & 23\,x_1 & + & 32\,x_2 & & \\
 & 10\,x_1 & + & 6\,x_2 & \leq & 2500 \\
 & 5\,x_1 & + & 10\,x_2 & \leq & 2500 \\
 & 7\,x_1 & + & 7\,x_2 & \leq & 2050 \\
 & & & x_1 & \geq & 0 \\
 & & & x_2 & \geq & 0
\end{array}$$

It's MSM tableau is:

Current Basis	Nonbasic variables x_1	x_2	Basic variables x_3	x_4	x_5	Values of basic variables
x_3	10	6	1	0	0	2500
x_4	5	10	0	1	0	2000
x_5	7	7	0	0	1	2050
Criteria	-10	-20	0	0	0	0
Rows	-23	-32	0	0	0	0

Once a problem is represented in the MSM tableau, we apply a series of matrix transformations to find objective maximizations. In the criteria row, $z_{ij} < 0$ indicates that we can improve on the *i*th objective if we introduce the x_j into the basis; we improve z_{ij} for each unit increase in x_j. A basic variable x_j is introduced into the basis by pivoting about some y_{ij}. With such knowledge, we can search through MSM tableaus to generate all extreme points which maximize one or more criteria. Below is the MSM tableau after two transformations. Both criteria are maximized, since none of the z_{ij} are negative. Generally, some criteria are maximized while others are at suboptimal values. Thus, several (extreme point) solutions are typically required to represent all objective maximizations.

Current	Nonbasic variables	Basic variables	Values of basic			
Basis	x_1	x_2	x_3	x_4	x_5	variables
x_1	1	0	0.143	-0.086	0	185.714
x_2	0	1	-0.071	0.143	0	107.143
x_5	0	0	-0.5	-0.4	1	0
Criteria	0	0	0	2	0	4000
Rows	0	0	1	2.6	0	7700

The basic algorithm is a depth first search of multi-criteria simplex matricies containing unique bases. We attempt to derive a nondominate matrix by pivoting about all z_j columns ≤ 0, i.e., those which could lead to a nondominate matix. Once a unique nondominated matix is found, we introduce all z_j columns which are not dominated by another z_i column. In the previous two sentences "all" is computed depth first, i.e., we visit the first and store the rest. As we visit each matix (basis) we remove it from the list of bases to be visited. If several columns may serve as the pivot, we prefer the one which has the Min Σz_j; further, each new set of bases generated by the "all" statements are first sorted by Min Σz_j and then pushed onto the `unexplored_bases` list. Zeleny proves that the set of nondominated extreme points is finite and shows the algorithm will complete after a finite number iterations[93]. Below, we sketch out the general algorithm as we have implemented it.

```
(defun MCSM ()
  (if (feasible_solution)
      (loop do (create_new_tableau xi unexplore_bases)
               (if (unique xi)
                 (progn
                   (save_solution xi)
                   (MCSM_search_aux))
                 (if (less_than_0 zjs)         ;xi is dominated!
                     (add_to zjs unexplore_bases);Pivot may lead to new basis.
                     (progn
                       (if (nondominated xi) (save_solution xi))
                       (MCSM_search_aux))))
            until empty(unexplored_bases))))

(defun MCSM_search_aux ()
   (if (nondominated_pivot @zjs)
       ;;Exist pivots leading to a xi+1 which dominate all other xi+1?
       (add_to @zjs unexplore_bases)
       (if (and not_all_0s_zj (nondominated xi))
           ;;Store those bases which might lead to nondominated xi.
          ;;zj columns which are not zero and lead to unexplored bases.
          (add_to not_all_0s_zj unexplored_bases))))
```

Below, we also show the algorithms for determining dominance and uniqueness. The non-dominance algorithm applys the simplex method to the z_{ij} subtableau. It will acquaint the reader with the general simplex pivot method; a more complex form is used to pivot the MCSM tableau during `create_new_tableau`. The uniqueness algorithm is supplied for completeness. (See[92] for further details.) To determine dominance, we apply the single ciriteria linear simplex method to the z_{ij} subtableau. If the tableau can be maximized, then our x_i is nondominated. Phase I of two phase simplex method is unnecessary because the value column is all 0, i.e., there is no need to check if there is a negative value.

```
(defun NonDominated (Z_tableau)
  (loop for pivot = (find_pivot Z_tableau)
        until (null pivot)
        (if reached_max(pivot)
            (return t)
            (pivot Z_tableau pivot))))
```

To find a pivot, we search through the objective (W) row for negative elements (since we are maximizing). When one is found, search that column for the largest positive value in the nonbasic variables. If one is found, we have our pivot; otherwise, we must continue searching the objective row. If no pivot is found and there was a negative element in the objective row, then we have reached a maximun.

```
(defun Find_Pivot (tableau)
  (loop column = from 1 to nonbasics(tableau)
        for negative = (or negative (< tableau[W_row,column] 0))
        for row = (and negative (row_of_max_column_value column))
        do (if row (return (list row column))) ;Early termination
        finally (return negative)))
```

To pivot, replace each element in the pivot row (except the pivot) by itself divided by the pivot.. Every other element e, where e = tableau[r,c], is replaced by: -1 * (e - (pivot_row[r] * pivot_column[c]) / pivot).

```
(defun Pivot (tableau row column)
  (loop with pivot = tableau[row,column]             ;compute key row
        for col from 1 to columns(tableau)
        tableau[pivot_row,col] = (/ tableau[pivot_row,col] pivot))
  (loop for r from 1 to rows(tableau)         ;subtract & multiply rows
        for key_value = tableau[r,column]
        unless r = row
        (loop for c in columns
              (if (= c column)
                  tableau[r,c] = 0
                  tableau[r,c] = tableau[r,c] -
                                 (* key_value tableau[row,c]))))))
```

A unique tableau conatains a maximized objective and no other adjacent tableaus which also maximize it. This is determined by verifying the objective row, say j, has all values $z_j \geq 0$ and any values $z_j = 0$ have columns $z_j \geq 0$.

```
(defun Unique (tableau)
  (loop for r in objective_rows
     thereis (loop for c from 1 to columns(tableau)
                   always (and tableau[r,c] ≥ 0
                               (if tableau[r,c] = 0
                                   (loop for rr in objective_rows
                                         for val = tableau[rr,c]
                                         for minus = (or minus val < 0)
                                         ;;Early termination if positive val.
                                         do (if val ≥ 0) (return t)
                                         finally (return minus))
                                    t)))))
```

BIBLIOGRAPHY

1. R. Alterman, "Adaptive planning," *Cognitive Science* **12** (1988) 393-421.

2. J.S. Anderson and A.M. Farley, "Plan abstraction based on operator generalization," *Proceedings of AAAI*, (August 21-26 1988) 100-104.

3. J.S. Anderson and A.M. Farley, "Partial commitment in plan composition," CIS-TR-90-11, Univerisity of Oregon (1990).

4. American Library Association, *Circulation policies of academic libraries in the United States, 1968*, American Library Association(1970).

5. M.H. Bazerman, *Judgment in managerial decision making*, John Wiley & Sons(1986).

6. L. Beck and T. Perkins, "A survey of software engineering practice: tools, methods, and results," *Transactions on Software Engineering* SE-9 (September 1983) 541-561.

7. V. Berzins, "On merging software extensions," *Acta Information* **23** (1986) 607-619.

8. D.G. Bobrow, *Qualitative reasoning about physical systems*, The MIT Press, Cambridge, Massachusetts(1985).

9. T.X. Bui, *Co-oP: A group decision support system for cooperative multiple criteria group decision making*, Springer-Verlag(1987).

10. B.R. Burkhalter and P.A. Race, "An analysis of renewals, overdues, and other factors influencing the optimal charge-out period," in: Eds. B.R. Burkhalter, *Case studies in systems analysis in a university library*, The Scarecrow Press, Inc. , Metuchen, N.J. (1968) 11-33.

11. R.M. Burstall and J.A. Goguen, "Putting theories together to make specifications," *Proceedings of the 5th IJCAI*, (1977) 1045-1058.

12. M.H. Burstein, "Concept formation by incremental analogical reasoning and debugging," in: Eds. R.S. Michalski, J.G. Carbonell, T.M. Mitchell, *Machine learning: an artificial intelligence approach*, Tioga Publishing (1986) 351-369.

13. D.N. Card, F.E. McGarry, and G.T. Page, "Evaluating software engineering technologies," *Transactions on software engineering* **SE-13** (July 1987) 845-851.

14. D. Chapman, "Planning for conjunctive goals," *Artificial Intelligence* **32** (1987) 333-377.

15. J. Conklin, "Interissue dependencies in gIBIS," STP-091-89, MCC (February 10, 1989).

16. S.E. Conry, R.A. Meyer, and V.R. Lesser, "Multistage negotiation in distributed planning," in: Eds. A.H. Bond, L. Gasser, *Readings in distributed artificial intelligence*, Morgan Kaufmann , San Meteo, California (1988) 367-384.

17. B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *CACM* **31** (November 1988) 1268-1287.

18. Randall Davis and Reid G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence* **20** (1983) 63-109.

19. T. DeMarco, *Structured analysis and system specification*, Yourdon(1978).

20. M. Deutsch, *The resolution of conflict: constructive and destructive processes*, Yale University, New Haven(1973).

21. K. Downing and S. Fickas, "Specification criticism via policy-directed envisionment," CIS-TR-90-05, University of Oregon (February 27, 1990).

22. S. Easterbrook, *Elicitation of requirements from multiple perspectives*, Imperial College of Science, Technology and Medicine, London(May 1991).

23. M.S. Feather, "Language support for the specification and development of composite systems," *Transactions on Programming Languages and Systems* **9** (April 1987) 198-234.

24. M. S. Feather, "Constructing specifications by combining parallel elaborations," *Transactions on Software Engineering* **15** (February 1989) To appear (Also available as RS-88-216 from ISI).

25. L. Festinger, *Conflict, Decision, and Dissonance*, Tavistock Publications, Ltd., London(1964).

26. S. Fickas, "Automating the transformational development of software," *Transactions on Software Engineering* **SE-11** (November 1985) 1268-1277.

27. S. Fickas, "A knowledge-based approach to specification acquisition and construction," CIS-TR-85-13, University of Oregon (November 1985).

28. S. Fickas and John Anderson, "A proposed perspective shift: viewing specification design as a planning problem," *Fifth international workshop on software specification and design*, (May 19-20, 1989) 177-184.

29. S. Fickas, J. Anderson, and W.N. Robinson, "Formalizing and automating requirements engineering," CIS-TR-90-03, University of Oregon (April 6, 1990).

30. S. Fickas, J. Anderson, and W. Robinson, "The KATE project: supporting specification construction," CIS-TR-90-24, University of Oregon (December, 1990).

31. D. Gentner, "The mechanisms of analogical learning," in: *{???}*, (198?) 199-241.

32. Michael P. Georgeff, "Communication and interaction in multiagent planning," *Proceedings of 1983 conference of the AAAI*, (1983) 125-129.

33. J.A. Goguen, "Reusing and interconnecting software components," *Computer* **19** (February 1986) 16-28.

34. R. Greiner and D.B. Lenat, "A representation language language," *Proceedings*, (1980) 165-169.

35. R.P. Hall, "Understanding analogical reasoning: computational approaches," 86-11, University of California, Irvine (June 4, 1986).

36. K.J. Hammond, "Learning to anticipate and avoid planning problems through the explanation of failures," *AAAI*, (1986) 556-560.

37. C. Hayes, "Using goal interactions to guide planning," *AAAI-87*, (1987) 224-228.

38. S. Horwitz, J. Prins, and T. Reps, "Integrating non-interfering versions of programs," #690, University of Wisconsin-Madison (March 1987).

39. M. Inuiguchi, H. Ichihashi, and H. Tanaka, "Fuzzy programming: a survey of recent development," in: Eds. R. Slowinski, J. Teghem, *Stochastic versus fuzzy approaches to multiobjective mathematical programming under uncertainty*, Kluwer academic publishers (1991) 45-68.

40. I.L. Janis and L. Mann, *Decision making : a psychological analysis of conflict, choice, and commitment*, The Free Press, New York(1979).

41. E. Jantsch, *Technological Forecasting in Perspective*, Organization for Economic Co-operation and Development, Paris(1967).

42. W.L. Johnson, "Specification as formalization and transformating domain knowledge," in: Eds. M. Lowry, R. McCartney, D. Smith, *Proceedings of the workshop on automating software design*, AAAI (August 25, 1988) 48-55.

43. G. Kaiser, "Composing software systems from reusable building blocks," *Twentieth Hawaii International Conference on Systems Sciences*, (January 1987)

44. S. Kedar-Cabelli, "Purpose-directed analogy," in: *Proceedings of the International Conference of the Cognitive Society*, (1985) 150-159.

45. R. Kemmerer, "Testing formal specifications to detect design errors," *Transactions on Software Engineering* **SE-11** (January 1985) 32-43.

46. Klee, "Unknown???," in: Eds. G.B. Dantzig, A.F. Veinott, Jr., *Mathematics of the decision sciences: part 1*, American mathematical Soceity , Providence, RI (1968)

47. M. Klein and S. C-Y Lu, "Run-time conflict resolution in cooperative design," *AI and Design Workshop*, (1988) To appear.

48. R. Kling and W. Scacchi, "The web of computing: computer technology as social organization," in: Eds. M. Yovits, *Advances in Computing*, Academic Press, Inc. (1982) 1-90.

49. D.F. Kohl, *Circulation, interlibrary loan, patron use, and collection maintenance: A handbook for library management*, ABC-Clio Inc.(1986).

50. K.L. Kraemer and J.L. King, "Computer-based systems for cooperative work and group decision making," *Computing Surveys* **20** (June 1988) 115-146.

51. D.B. Lenat, "The nature of heuristics," *Artificial Intelligence* **19** (1982) 189-249.

52. D.B. Lenat, "EURISKO: A program that learns new heuristics and domain concepts: the nature of heuristics III," *Artificial Intelligence* **21** (1983) 61-98.

53. D.B. Lenat, "Theory formation by heuristic search: the nature of heuristics II," *Artificial Intelligence* **21** (1983) 31-59.

54. D.B. Lenat and J.S. Brown, "Why AM and Eurisko appear to work," *Proceedings of AAAI*, (1985) 236-240.

55. Association of Research Libraries, "Collection development policies 1977," *Systems and Procedures Exchange Center*, (November 1977)

56. Association of Research Libraries, "Automated circulation," *Systems and Procedures Exchange Center*, (April 1978)

57. Association of Research Libraries, "SPEC kit on goals and objectives 1979," *Systems and Procedures Exchange Center*, (October 1979)

58. B.P. Lientz, "Issues in software maintenance," *Computing Surveys* **15** (September 1983) 271-278.

59. Marc Luria, "Goal conflict concerns," *IJCAI-87*, (1987) 1025-1031.

60. D. McDermott, "Planning and acting," *Cognitive Science* **2** (1978) 71-109.

61. E. Mumford and D. Henshall, *Aparticipative approach to computer systems design*, Halsted Press, New York(1979).

62. E. Mumford and M. Weir, *Computer systems in work design—the ETHICS method*, Associated Business Press, London(1979).

63. J. M. Neighbors, "The Draco approach to constructing software from reusable components," *Transactions on Software Engineering* **SE-10** (September 1984) 564-574.

64. N.J. Nilsson, *Principles of artificial intelligence*, Tioga, Palo Alto, CA(1980).

65. J.W. Perkins and P.N. Clingen, *Inglewood public library circulation procedures*, Inglewood public library(1972).

66. D.G. Pruitt, *Negotiation Behavior*, Academic Press Inc.(1981).

67. Howard Raiffa, *The art and science of negotiation*, Harvard University Press(1982).

68. C. Rich, R.C. Waters, and H.B. Reubenstein, "Toward a requirements apprentice," *4th International workshop on software specification and design*, (April 3-4, 1987) 79-86.

69. C. Rich and R.C. Waters, "The programmer's apprentice: a research overview," *Computer*, (November 1988) 10-25.

70. C. Rich and R.C. Waters, *The programmer's apprentice*, ACM press, New York(1990).

71. S.P. Robbins, *Organizational behavior: concepts, controversies, and applications*, Prentice Hall, NJ(1983).

72. W.N. Robinson, *Towards formalization of specification design*, Masters thesis, University of Oregon(June 1987).

73. W.N. Robinson, "Automating the parallel elaboration of specifications: preliminary findings," Technical Report CIS-TR-89-02, University of Oregon (February 1989).

74. W.N. Robinson, "Integrating multiple specifications using domain goals," *5th International workshop on software specification and design*, (1989) 219-226 (Also available as CIS-TR-89-03 from the University of Oregon).

75. D.T. Ross and K.E. Schoman Jr., "Structured analysis for requirements definition," *Transactions on Software Engineering* SE-3 (January 1977) 6-15.

76. G.R. Salanick and J.F. Porac, "Distilled ideologies: values derived from causal reasoning in complex environments," in: Eds. H.P. Sims, Jr., D.A. Gioia, and Associates, *The thinking organization: dynamics of organizational social cognition*, Jossey-Bass Publishers (1986) 75-101.

77. W. Scacchi, "Managing software engineering projects: a social analysis," *Transactions on software engineering* **SE-10** (January 1984) 49-59.

78. H.R. Schwarz, *Numberical analysis: a comprehensive introduction*, John Wiley & Sons(1989).

79. M.L.G. Shaw and B.R. Gaines, "A methdology for recognizing consensus, correspondence, conflict and contrast in a knowledge acquisition system," in: *Workshop on knowledge acquisition for knowledge-based systems*, , Banff (November 7-11, 1988)

80. R. Slowinski and J. Teghem, *Stochastic versus fuzzy approaches to multiobjective mathematical programming under uncertainty*, Kluwer academic publishers(1991).

81. W. Swartout and R. Balzer, "On the inevitable intertwining of specification and implementation," *CACM* **25** (1982) 438-440.

82. Katia Sycara, "Resolving goal conflicts via negotiation," *Proceedings of the AAAI-88*, (1988) 245-250.

83. K.P. Sycara, "Resolving adversarial conflicts: an approach integrating case-based and analytic methods," GIT-ICS-87/26, Georgia Institute of Technology (1987).

84. I. Vessey and R. Weber, "Some factors affecting program repair maintenance: an empirical study," *CACM* **26** (February 1983) 128-134.

85. D.M. Weiss and V.R. Basili, "Evaluating software developments by analysis of changes: some data from the software engineering laboratory," *Transactions on Software Engineering* **SE-11** (February 1985) 157-168.

86. K.J. Werkman, "Knowledge-based model of using shareable perspectives," *Proceedings tenth international conference on distributed artificial intelligence*, (October 1990) 1-23.

87. R. Wilensky, *Planning and understanding*, Addison-Wesley(1983).

88. D.E. Wilkins, "Domain-independent planning: representation and plan generation," *Artificial Intelligence* **22** (1984) 269-301.

89. J.M. Wing, "A study of 12 specifications of the library problem," *Software*, (July, 1988) 66-76.

90. D. von Winterfeldt and W. Edwards, *Decision analysis and behavioral research*, Cambridge Univerisity Press(1986).

91. P.L. Yu, *Multiple-criteria decision making*, Plenum Press, New York(1985).

92. Milan Zeleny, *Multiple criteria decision making*, McGraw-Hill(1982).

93. M. Zeleny, *Linear multiobjective programming*, Springer-Verlag, New York(1974).

94. M. Zelkowitz, R. Yeh, R. Hamlet, J. Gannon, and V. Basili, "Software engineering practices in the U.S. and Japan," *Computer* 17 (June 1984) 57-66.