Negotiation Freedoms for Requirements Engineering

William N. Robinson
Stephen Fickas

CIS-TR-90-04
April 6, 1990

Abstract

Requirements and specification acquisition are intertwined processes of conflict management. Within an individual, or within a group, conflicting requirements must be reconciled and specified. We propose three freedoms to aid requirements and specification acquisition: freedom of preference, freedom of conflict, and freedom of compromise. These freedoms imply the need for multi-agent requirements, multi-agent specifications, and requirements negotiation. We have incorporated these freedoms and their implications into our model of requirements engineering. Requirements are negotiated in the context of combining multiple specifications. This process is aided by integrative reasoning, our computer counterpart to a type of negotiation behavior. By providing the analyst with computer-aided-negotiation methods, we directly support the intertwined processes of requirements and specification acquisition.

Department of Computer and Information Science
University of Oregon

Negotiation Freedoms for Requirements Engineering

William N. Robinson

Stephen Fickas

Department of Computer and Information Science, University of Oregon, Eugene, OR, 97403, U.S.A.

Send comments to robinson@cs.uoregon.edu

ABSTRACT

Requirements and specification acquisition are intertwined processes of conflict management. Within an individual, or within a group, conflicting requirements must be reconciled and specified. We propose three freedoms to aid requirements and specification acquisition: freedom of preference, freedom of conflict, and freedom of compromise. These freedoms imply the need for multi-agent requirements, multi-agent specifications, and requirements negotiation. We have incorporated these freedoms and their implications into our model of requirements engineering. Requirements are negotiated in the context of combining multiple specifications. This process is aided by intergrative reasoning, our computer counterpart to a type of negotiation behavior. By providing the analyst with computer-aided-negotiation methods, we directly support the intertwined processes of requirements and specification acquisition.

---

1. Introduction

Typically, requirements acquisition consists of eliciting "correct" and "consistent" descriptions of user desires. Many methodologies, languages, and tools support this paradigm. Unfortunately, while simplifying automation, this paradigm binds the hands of the analyst.

Analysts require freedoms. Freedom from implementation, completeness, and consistency[3, 36]. Analysts need to specify *what* is required, without speaking of *how* to do it. Now, analysts require a new freedom: the freedom to negotiate.

A single user can express conflicting and inconsistent requirements. Multiple users exacerbate this problem by disagreeing about requirements. Aiding their negotiation has been outside of software engineering. Consequently, user decisions and rationale are lost while their effects live on. Users are left to wonder about strange "features". Worse still, maintainers may change features without understanding the (possibly dire) outcome.

We strive to overcome these problems by modeling and assisting requirements negotiation. Specifically, we address

- *acquiring preferences*, and
- *managing conflicts.*

By doing so, we support the process of deciding what to specify while recording the rationale for future use.

Analysts need the freedom to disagree. Having incorporated aspects of negotiation into a specification methodology, we propose three negotiation freedoms for acquisition:

(1) **Freedom of Preference**

(2) **Freedom of Conflict**

(3) **Freedom of Compromise**

These freedoms have three direct implications:

(1) *Multiple Requirements*

(2) *Multiple Specifications*

(3) *Specification Negotiation*

We argue for these freedoms and their implications using theories from both computer science and the Social and Decision Sciences. We illustrate their usefulness with an example.

We present a negotiation design method. It combines domain modeling with specification analysis. Various user perspectives of requirements are represented. Specifications are derived for each perspective. Their components support user requirements while exhibiting domain interactions. Through specification integration, requirements are negotiated in the system context. This approach addresses the intertwining of conflicting requirements, conflicting specifications, and their negotiation.

In this paper, we use a transcript excerpt to illustrate problems of conflicting user requirements (§ 2). Related research follows (§ 3). Then, our Multiple Perspective Specification Design (MPSD) methodology is presented (§ 4); it uses negotiation freedoms to aid resolution of conflicting requirements. Automated support for MPSD (§ 5) and an automated example (§ 6) follow. Finally, we conclude that requirements and specification acquisition are intertwined processes of conflict management (§ 7).

2. Acquiring Requirements

Three major tasks of requirements acquisition are: (1) acquiring user goals, (2) specifying those goals, and (3) integrating divergent goals.[1] Consider the following transcript of an initial dialogue between a systems analysts (A), a manager (M), and a user (U). They are speaking of an admissions management system.

[1]Goals are flexible requirements. Unlike constraints, they allow for partial satisfaction. This type of requirement is needed for specification negotiation (§ 4.3). In this paper, requirements are comprised of goals and constraints.

79 U As it is now, the variety at least, I get to take my eyes away from the computer. I'm spending half of my time writing letters on the computer, and to spend the other half putting information into the computer, ...

413 U Is the purpose to eliminate the card file?

414 A Yes. Yes.

415 U Then I'll just restate my original objection...

416 A Right.

417 U ...of sitting in front of the computer all day and entering this information is exceedingly boring...

418 A Yeah.

419 U ...and very hard on the eyes...

420 A Right. Well, for one thing you wouldn't be entering the information. And I admit that there's no way around looking at a computer screen if this is automated. So, if that's real distasteful to you, then that could be a problem.

421 M Weren't you talking about having the computer print cards with all the information...

422 A That's true.

423 M Or, just generate a hard copy so that you have some backup.

424 A That's true, but I guess we'd have to think about it more, it seems like the best reasons for doing this–ah, that's true, maybe a card file would be useful. It seems to me that a person that likes to use computers, one of the best reasons to have this automated is because it's a lot easier to get access to information rather than looking through the log file or card file.

425 U I think I find it personally easier to pull out a card file, to get the name alphabetically than it is to punch it into the computer and wait for it to come...

426 A Yeah, right.

The analyst, A, supports typical goals of an MIS system, e.g., automated filing. However, U has other goals. She desires to work away from a computer at a variety of tasks. Perceiving the specified system as conflicting with her goals, she negotiates. At issue is the amount of computing time in which U must engage. She trys to reduce her computer time by stressing the need for a card file. The analyst gives in. Interactions 424-426 exhibit the compromise: a card file will complement the automated system.

The above interaction was typical in acquisition protocols we conducted[19]. Conflicts arose mainly from the pursuit of goals, not because of mistakes. Such conversations are similar to contract negotiations: parties interact to gain a mutually beneficial outcome. Unlike contract negotiation, requirements negotiation is rarely investigated or even acknowledged.

Once goals are acquired, they must be met in a specification. Complex specification involves negotiation amongst a group of analysts[4, 8]. Each analyst represents a user's concern or a domain of knowledge. These concerns may be initially sketched out in personal specifications. Later, they are integrated into the delivered specification.

Often, one individual stands above the rest with skills of: interdisciplinary information integration, communication, and motivation. In one study, such individuals had knowledge which "...allowed them to integrate different, sometimes competing, perspectives on the development process."–p.1271[8]. Such analysts serve the role of arbitrator or mediator of conflicts. But, while they do apply negotiation knowledge, they do so without support. Analysts have "...lamented having no tools for capturing issues and tracking their status...Failure to resolve issues frequently did not become obvious until integration testing."–p. 1278[8].

Unfortunately, requirements negotiation remains largely unexplored. Next, we discuss work to date.

3. Related Research

Until recently, very little research has directly addressed the presence of negotiation behavior in requirements acquisition. Even those that have, neglect to addressed negotiation subprocesses[5, 20]. However, negotiation is generally recognized to exist; Ross typifies its early treatment.

> To succeed, the task of the analysis must be properly managed and coordinated, and the requirements definition effort must embody multiple viewpoints. These viewpoints may be overlapping and, occasionally contradictory–p. 10[49].

Later, Scacchi elevated the importance of negotiations.

> Problems found in specifications may be due to oversights in their preparation or conflicts between participants over how they believe the system should function...Each of these questions point to tacit or explicit negotiations between participants that must occur in the course of getting system specifications developed. Subsequently, the outcome of these negotiations will shape how the specifications will be.–p. 54[51].

Recently, a survey of large systems design concluded, "...developing large software systems must be treated, at least in part, as a learning, communication, and negotiation process."–p. 1282[8]. However, only now are empirical and modeling studies of specification concerned directly with negotiation.

Bendifallah and Scacchi[5] observed five student teams building similar software specifications over a ten day period. As part of their analysis, they hypothesized six categories of specification behavior. Three of these categories are particularly relevant: separating a problem into sub-tasks, resolving conflicts, and integrating results. This is encouraging and supportive of our methodology. However, we require a more formal and detailed representation than that given by Bendifallah and Scacchi to build specification design tools.

Finkelstein and Fuks view specification design as a multi-party negotiation problem[20]. They have developed a formal model of negotiation dialogue. Constrained by dialogue rules, knowledge sources remove inconsistent beliefs through communication. This view of specification as a multi-agent communication task is encouraging. However, such protocol oriented models speak to only a narrow aspect of negotiation. They avoid identifying conflicts, representing conflicts, and generating resolutions; these are basic concepts of negotiation.

Our research concerning negotiation originated with an automated assistant, **Oz**, which embodied Feather's *parallel elaboration* specification methodology[44]. Simply stated, his methodology calls for independent development of separable functional specification aspects[16, 17]. With it, independently developed designs are not constrained to have consistent interfaces. While this simplifies design, it complicates design integration. In our approach, integration is assisted through negotiation techniques[15, 45, 46].

We view specification design as an interplay between the acquisition of user goals and their representation in a specification language. The first process entails formalizing what users want to achieve (requirements acquisition); the second entails formalizing how their needs can be met (specification acquisition).[2]

Only through operational prototypes can goals be *discovered* to interact.[3] A *prototype* is a proposed structure satisfying a variety of goals; its components support goals and exhibit interactions. Where a complete model of goal interactions is lacking, analysis of prototypes is essential. Specifications serve this role in requirements analysis.

[2]Such specifications do not (generally) include software design or implementation decisions. They describe relations of the system and its environment[3].

[3]Goal interaction descriptions, such as probabilities of competition[10] or interaction-resolution pairs[56], are useful. But, they combinatorically increase with the number of goals[37] and depend on ever changing technology[26, 63].

As a specification is created, it will become apparent that some goals have representations which interfere while others do not. Their interactions drive designers to seek alternative representations, relax goals, and drop goals[11]. This is conflict management, a process of negotiation.

Our approach can be viewed as one of planning: establish user goals and attempt to find a plan that will map the goals into an operational form. However, it differs from traditional planning techniques in important ways:

(1) Instead of fixed state descriptions, our goals allow for partial satisfaction through preferences.

(2) Instead of a single agent, we take into account multiple agents, each with its own goal perspective.[4]

(3) Instead of constructing a single plan, we construct a plan for each agent (i.e., for each perspective) and integrate the plans into a unified whole.

The MPSD model addresses two types of conflict: conflicts within a single agent and conflicts among multiple agents. Multi-agent planning research has mainly focused on conflicts within a single plan. Conflicts are resolved by coordination using specialized protocols[7, 9, 12, 13, 23, 53].

Conflicts between agents with different goals (e.g., agent-1 wants X and agent-2 wants not-X) requires another type of knowledge: conflict resolution knowledge. Multi-perspective research has focused on general architectures[25, 31] and conflict resolution knowledge. Multi-perspective resolution knowledge generalizes the resolution methods of traditional single-agent planners[50, 54, 60] by directly addressing conflict among competing concerns[29, 32, 56, 57].

We too are focused on multi-perspective negotiation. Our architecture generalizes those above in its combination of analytic search, heuristic improvement, and case-based reasoning. (We also introduce a new method of heuristic improvement called the *Search for Compensation.*) MPSD derives its generality from its use of integrative reasoning. Like[29, 32] we test our negotiation model on design problems, specifically specification design. The context in which it is applied is presented next.

4. Multiple Perspective Specification Design

In the following subsections, we present negotiation freedoms, our specification design methodology, and supporting arguments. Section 5 presents automation of MPSD. There, more specific arguments are made concerning our model and conflict management techniques.

4.1. Negotiation Freedoms

Using current specification methodologies, analysts cannot represent and negotiate conflicting requirements. Yet, analysts must be free to represent preferences, conflicts, and compromise[39].

[4]We use the term *agent* to represent a class of users with similar concerns; multi-agent means multiple concerns (henceforth, *multi-perspective*). In contrast, AI planners view an agent as an entity or task to be controlled; multi-agent means multi-tasking (i.e., coordinating or scheduling multiple entities or tasks to achieve a goal set *from a single perspective*)[23]. Further, in planning and distributed problem solving[6] agents are often expected to cooperate to achieve common goals[48]. While we do assume a limited amount of cooperation, our agents are encouraged to state selfish needs; they are not constrained to have common goals.

Freedom of Preference

Some requirements are more important than others. In our model, goals represent requirements. Thus, *goal preferences* must be expressible. Also, there can be alternate means to specify a goal: *implementation preferences* must be expressible.[5]

Freedom of Conflict

Conflicts of substance occur during specification. These are not just syntactic mistakes, but reflect variant semantics. Also, conflicts can arise from inconsistent preferences. Goal, preference, and specification conflicts (and their resolutions) must be expressible.

Freedom of Compromise

When conflicts arise, users must be able to reduce their demands or accept substitute satisfaction; they must be able to delay their commitment[20]. Analysts must be free to derive and represent compromised goals.

Together, these freedoms have direct implications:

Multiple Requirements

Preference conflicts, concerning both goals and their specification, give rise to multiple perspectives of requirements. For example, user groups may internally agree on goals, but disagree between groups. To address their negotiation, it is useful to consider conflicting goals as interactions among alternative requirements perspectives.

Multiple Specifications

Multiple specifications are implied by both (1) alternative goal implementations and (2) variant requirements perspectives: users need to compare various specifications of a goal, and user groups need to represent their variant requirements perspectives in specifications. Specification conflicts are the conflicting means of variant perspectives.

Specification Negotiation

Conflicts imply negotiation. In MPSD, conflicts are detected between specifications, negotiated between goals, and implemented back in specifications. We use specifications to detect many goal interactions. Negotiation could take place solely in the goal space if all interactions were known. But, if we had such complete knowledge, the specification process would be of a different kind, i.e., akin to an application generator.

Our MPSD research concerns incorporating these freedoms into a specification methodology and providing automated support.

4.2. MPSD

Our Multiple Perspective Specification Design (MPSD) methodology entails (1) a domain model, (2) user perspectives, (3) specifications, and (4) integration.

Domain Model

The *domain model* is our requirements language. It represents knowledge of goals and their implementations. Also, some types of goal interactions are represented, e.g., *inherent conflict.*

Perspectives

Perspectives represent user motives. They consist of goals and constraints derived from the domain model. The derivation entails describing goal and implementation

[5]In this paper, *implementation* refers to (possibly abstract) means of specifying goal satisfaction; not to be confused with software implementation.

preferences.

Specifications

Specifications are perspective prototypes. Their components *implement* the goals described in perspectives. Through their operationality, specification components reflect relations implicit amongst the goals they support.

Integration

Integration is the process by which specifications are combined and goals are negotiated. Differences detected between specifications reflect possible conflicts between perspectives. Alternative goal implementations can sometimes remove these differences. However, they often are the result of goal conflicts. These conflicts can be resolved through *integrative reasoning* and then respecified. These techniques are illustrated in sections 5 and 6.

In sum, analysts represent the goals of various user groups in perspectives and derive supporting specifications. The specifications are integrated, during which conflicting goals are negotiated. Finally, their resolution is specified. The result is a multi-agent negotiated specification.

As an example, recall the conflict of section 2. Using MPSD, analysts can independently represent the perspectives of the manager and the user. Next, specifications are derived from the perspectives. These specifications represent "ideal" systems in that they exclude the needs of other agents. For example, U's specification only has an indexed card file system; A's only has an on-line database. Specification integration follows: differences are noted; conflicting goals are determined; resolutions are proposed; and a resolution is accepted. In the example, the result was both a card file and a computer system.

4.3. Supporting Structures

We support negotiation freedoms by modeling integrative behavior. Integrative behavior is cooperative behavior observed in many profitable negotiations. The following substructures support requirements negotiation. They do so by taking part in *integrative reasoning*, the computer counterpart of integrative behavior. The next subsection places them in the context of integrative behavior.

Goals

Goals are nonoperational descriptors of achievement in a domain[38]. They are linked together by their implementation relations. For example, the goals LoanPeriod, MaterialLimit, and Renewal are linked under the goal WorkingSet because they are implementations of it.[6] (The leaves of this goal graph can be operational specification segments.)

Goal Preferences

Some goals may be inherently preferable over others, e.g., Life over Death. When such goal preferences are expressed in a domain model or perspective, it constrains all implementation preferences to bear the same order; i.e., Life will always be preferred over Death in the implementation of any goal.

Implementation Preferences

An implementation preference orders implementations relative to the satisfaction of a particular goal. For example, a Wage implementation preference could be

[6]Working set is a library science concept. To support research, a library must allow patrons to use a set of materials for sufficient time. This can be achieved through loan periods, material renewal, and multiple material borrowing[30].

($1,$2,...$1000,...) with acceptance increasing with dollar value. However, employers may have an inverted preference: (...$1000,...,$2,$1)! Two goals that are ordered the same in all implementation preferences can be placed in a goal preference relation. Analysts use implementation preferences to try to maximize satisfaction of user perspectives. They aid in the evaluation of substitute and compromise conflict resolutions.

Ideal Perspectives
The more *ideal* initial perspectives are (coupled with cooperation), the higher the resulting achievement. Implementation preferences describe ranges of achievement. Through their combination, implementation preferences define a multi-dimensional search space of resolutions. Setting low aspirations reduces the search dimensions: compromising early can lead to suboptimal achievement.

Goal Prototypes: specifications
While goals are effective descriptors of needs, they are inadequate for interaction analysis. Interaction knowledge must be known *a priori* or gleaned from linked prototypes. Specifications are effective prototypes. Through analysis of operational components, interactions can be detected and traced to goals.

Preference Feedback
Choosing a resolution from a set of alternatives can be difficult. Considering alternatives by their satisfaction of goal can help. However, people still have difficulty understanding goal interactions and comparing alternatives. They exhibit intransitive preferences: first, preferring A over B over C; then, B over C over A. Such decisions are context dependent. However, an interactive procedure which provides feedback of goal interaction can help. Such a procedure can aid an analyst in deriving preferences over user implementation preferences. These *implementation meta-preferences* are used to choose a final resolution.

These structures were derived from our attempt to automate integrative behavior. This behavior and the theory of its support is introduced next.

4.4. Negotiation Requirements

Productive *integrative bargaining* is more likely to occur in negotiations involving cooperative parties employing a *strategy of flexible rigidity*[42]. This strategy consists of incorporation, information exchange, and search. *Incorporation* is the act of augmenting a proposal with some element of an opposing, previously made, proposal. *Information exchanges* are communications which provide insight into another party's motivational structure (goals, values and constraints). *Search* involves jointly considering a variety of proposals. Parties present them based entirely on their own perspective with little consideration as to why others favored or rejected previous proposals. Each of these individual aspects, and the strategy of flexible rigidity as a whole, have substantial empirical support[42].

We have mapped the *strategy of flexible rigidity* onto specification design. Proposals are specifications; motivational structures are perspectives; search and information exchange are part of integration. Parts of the strategy are automated through *integrative reasoning* (§ 5), our model of integrative behavior. Before its presentation, we justify its use.

4.4.1. Implementation Preferences

Integrative agreements can only be achieved by communication of *real needs*[59]. Studies conducted or reviewed by Pruitt agree[42]. Most situations benefit from the exchange of preference and numeric information, be it explicit or implicit. More detailed information exchange involving "goals, values, and priorities is theoretically capable of transmitting rich information from which integrative formulas can be devised. However our data suggest that it has limitations..."–p. 171[42]. Pruitt postulates these difficulties are due to: parties uncertain of their own motives, lack of trust by listeners, and poor interpretation of statements by listeners.

Zeleny proposes implementation preferences as the basic expression of needs. Preferences "represent *directions* of improvement or preference along individual attributes or complexes of attributes"–p. 15[63]. These flexible goals allow for maximal achievement, since they don't pick a value to attain *a priori*. If preferences are unavailable when goals conflict, resolution evaluation is impossible. For example, one needs to know that a $14,000 car may be preferred over a $16,000 car if a $15,000 car can't be had; one prefers to minimize expenses. Preferences are particularly useful in domains where many goal interactions are unknown, and consequently conflicts are common. Requirements acquisition is such a domain.

Implementation preferences are used two ways. First, to evaluate achievement. When goals interfere, feasible implementations may only support submaximal achievement. Such implementations are evaluated through preferences. This helps guide resolution search.

Interpolation is a second way preferences are used. Alternative implementations can be plotted in a n-dimensional preference space. Each axis is a preference scale. Alternatives are positioned along each scale based on their preferred goal implementation status. New alternatives can be sought between existing preferences. Nondominated alternatives can be identified. These ideas aid resolution search.

4.4.2. Goal Perspectives

Zeleny's *displaced ideal* theory defines a search space of resolutions and an ideal to obtain[63]. Alternatives are defined by their ability to satisfy a goal based on implementation preferences. These preferences form the dimensions of a resolution search space. The *ideal* is the (infeasible) composite of each preference's maximum achievable value within the feasible alternatives, i.e., the best known implementation for every goal without any of the negative interactions.

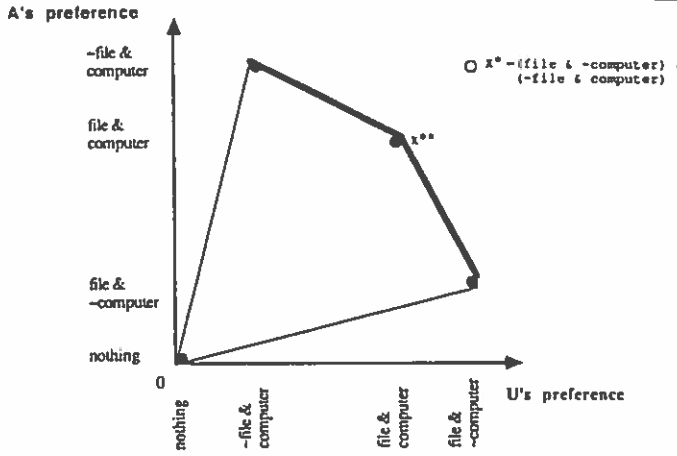

Figure 1. Searching for the Ideal Resolution.

Figure 1 illustrates Zeleny's ideal using interactions 79-426 from § 2. From the protocol, we infer that U desires (1) *only a card file and no computer*. But, A desires (2) *only a computer and no card file*. The composite infeasible ideal, depicted x^*, includes both statements 1 and 2. Through search, x^* is displaced in favor of the feasible ideal, x^{**}. The three remaining points are other feasible alternatives. Together, the feasible alternatives circumscribe the space of possible, yet undescribed, alternatives. The bold line represents compromise alternatives. Such alternatives are *nondominated* since no alternative is between them and the ideal, x^*. Despite being infeasible, describing the ideal is useful.

The search for the displace ideal is based both on moving away from an anti-ideal (the worst of all issues) and moving toward the ideal.

> 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 [goals]. Such displacements induce changes in evaluations, attribute [goal] importance, and ultimately in the preference ordering of the remaining alternatives. – p. 143[63].

Search starts at the anti-ideal and moves toward the ideal. Near the ideal there is a set of *nondominated alternatives*; these alternatives are equal distance from the ideal. One of these alternatives must be chosen (a compromise) or a negative interaction must be removed. Removing a negative interactions moves the nondominated set closer to the ideal. This method is presented in section 5.5.2.

Arrow's axiom, influential and controversial in decision theory, states that *only feasible alternatives* have influence on a rational decision[2]; infeasible ideal resolutions should not be considered. It implies that perspectives should be unified before specification construction begins; that ideal prototypes need not be built. However, while attempting to verify Arrow's axiom experimentally, Festinger and Walster obtained evidence to the contrary[18]. Infeasible alternatives do have an influence on preferences. Thus, specifications should be attached to perspectives early. They influence user preferences and, through their operationality, provide preference feedback. Specifying from multiple perspectives elucidates user goals. It helps requirements acquisition.

The strategy of combining extreme requirements perspectives assists in: (1) exploring specifications, (2) ranking specifications, (3) deriving maximally feasible specifications, and (4) predicting the existence of ideal specifications. This last case is of particular importance; the very description of the ideal alternative may point to its existence or suggest its achievement; cf. Zwicky's morphological analysis[26] or Lenat's discovery mechanism[33-35]. Our *dissolution heuristics* (§ 5.5.2) serve this role.

4.4.3. Preference Procedures

Often, many nondominated resolutions exist. One of two basic methods are typically used to choose a final resolution. Static analysis, such as utility analysis, can compute the best resolution. Or, a tool can interactively assist a decision agent (e.g., an arbitrator). These methods are considered in turn.

In utility analysis, agents define a function mapping implementations onto a scale[28]. Agents must also specify the mapping of goals onto a scale. Determining a compromise becomes simply the maximization of the scaled preference functions.

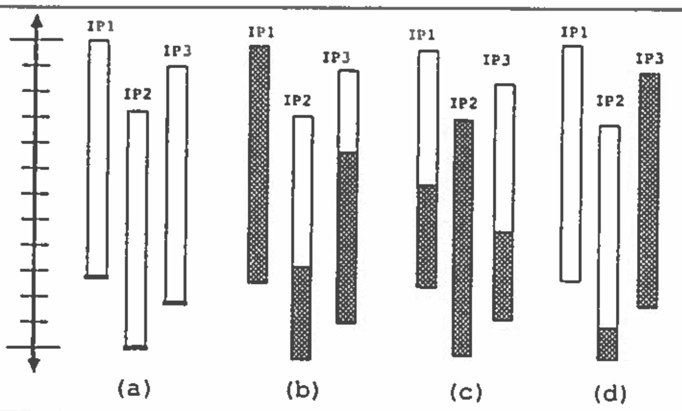

Figure 2. Choosing a Compromise with IDEA.

Unfortunately, utility theory is only relevant when people maintain consistency in their preferences; utility functions must be transitive–this is a direct result of the *additivity assumption*, i.e., the summing of weights[1]. There is evidence to the contrary; people display intransitive preferences[1] and are poor at combining relative strengths in a consistent manner[52]. Zeleny's theory explains this intransitivity as changes in the displaced ideal which causes a reordering of alternatives during resolution search[63]. He complements his search strategy with an interactive decision aid.

Zeleny's Interactive Decision Evolution Aid (IDEA)[63] assists the analyst in choosing a nondominated resolution. Here too, implementations must be mapped to a preference scale (see § 5.4). However, implementation meta-preferences are obtained through search. IDEA graphically displays user implementation preferences. The closer all potentials are together and to 100%, the better the alternative. (The ideal is achieved when all implementation preferences are at 100%). However, the analyst can choose an alternative that has significant variance in potentials reflecting trade-offs between user implementation preferences.

Initially, all potentials are at zero percent. The analyst moves toward the ideal by increasing potentials. Figure 2 illustrates this process. Interactions surface when increasing an implementation preference decreases another. For example, moving from (b) to (c) in figure 2 shows the trade-off between IP_1 and IP_2. Through this procedure, implementation meta-preferences are assigned.

Meta-preferences reflects a weighting between users. If one user has more authority than another, his preferences will be weighted more heavily. However, users typically have areas of authority. For example, a manager can have many of his preferences heavily weighted. Yet, a worker can have some of her preferences weighted more where her expertise applies. While one can model some preferences and meta-preferences *a priori*, many such decisions are context dependent[18]. This process of modeling agents, deriving meta-preferences, and choosing a compromise is called arbitration[43].

IDEA allows for intransitive preferences. It supports exploration of goal interactions and comparative analysis. Trade-offs are considered in the actual context, not *a priori*. Users can see the consequences of their preferences. Essentially, it's an interactive abstract prototyper of goal implementations, i.e., a requirements prototyper.

4.4.4. Summary

Integrative bargaining helps people derive beneficial solutions to their interacting needs. It consists of incorporating opposing proposals; communicating goals, values, and constraints; searching through alternatives; and intransitive preferences. These complex interacting behaviors result in high profits when individual participants are stubborn in their aspirations and seek innovative solutions. If participants seek an expedient resolution by abandoning goals early, lower satisfaction will result.

MPSD is the result of our effort to reap benefits of an integrative approach. MPSD representations and processes map onto the elements of integrative behavior. Through this correspondence, MPSD manages requirement conflicts and preferences. It frees the analyst to represent:

- Variant Goal Preferences
- Variant Implementation Preferences
- Implementation Meta-Preferences

These expressions allow freedom of preference, conflict, and compromise.

5. Automating MPSD

Having described MPSD, we now characterized its automation. Specifically, we present a generic integrator (GI). This facility unifies specifications through conflict management.

A GI has five general tasks to complete:

(1) *Correspondence Identification*

(2) *Conflict Detection*

(3) *Issue Formation*

(4) *Conflict Resolution*

(5) *Resolution Implementation*

and two major types of knowledge:

(1) *Domain Goal Knowledge*

(2) *Conflict Resolution Knowledge*

The following subsections describe the tasks in turn. Domain goal knowledge is introduced as part of the *Issue Formation* task; conflict resolution knowledge is introduced as part of the *Conflict Resolution* task.

5.1. Correspondence Identification

Correspondence identification determines which entities in two specifications denote the same object. This problem differs from graph isomorphism[21] since it involves a partial match; i.e., corresponding entities are not exactly equivalent. In fact, the power of integrative reasoning is derived from the differences between corresponding objects. Such differences can be combined via integration.

There are three basic ways in which correspondences can be determined. First, they can be given. Second, they can be determined from derivation records; correspondence means being similarly derived from a common ancestor[40]. Third, entities can be arbitrarily placed in correspondence, then ranked by the degree their associated structures match; e.g., the types of the corresponding objects, their substructures, and their relations with surrounding objects. The pairing structure with the highest accumulative score can be chosen[62].

The third method, constraint-directed determination, can be improved by a good initial guess and a good equivalence function. Initial correspondence can be established during specification retrieval (i.e., for reuse); or, based on a top-down model providing clues of possible correspondences (e.g., only place objects of the same type in correspondence)[61]. [7] Similarly, constraints obtained from domain knowledge or goal directed processing can assist in determining semantic or functional equivalence of specifications components. For example, functional knowledge of libraries can be used to determine the correspondence between a borrower and a patron, BORROWER≡ PATRON[14, 22, 27, 61, 62].

5.2. Conflict Detection

Once correspondences have been found, differences must be determined. Such differences are called *feature conflicts.* Feature conflicts can be combined through negotiation and placed in the integrated specification.

Conflicts are determined by feature comparisons between correspondences and amongst the remaining noncorresponding entities. These differences can be the product of the correspondence identification process or of direct comparison. After conflicts are determined, they must be characterized in a way which allows for their reconciliation. This is *issue formation.*

5.3. Issue Formation

An *issue* is a domain goal, recognized by all agents involved, as subsuming the direct interests each agent has in a conflict, and excluding extraneous interests of that conflict. We distinguish feature conflicts, which tend to imply mutual exclusion, from *issues* which exhibit a range of behaviors.

Consider a simple conflict concerning a book loan period. From a patron's perspective, loan periods should be as long as possible; this insures their ability to enjoy borrowed resources. On the other hand, a circulation librarian desires to insure equal access of resources to all patrons; hence, shorter loan periods provide higher turnover which enables greater access to a large population of patrons. If both parties expressed their preferences, we might observe the following feature conflict.

Patron:	"Loan periods should be six months!"
Librarian:	"Absolutely not! They must be two weeks!"

Above, "six months $\neq$ two weeks" is not the issue, Loan Period is the (most direct) issue[46]. Furthermore, by *issue formation* we mean the initial process by which issues are derived from the given conflicts. Any further processing used to bring in other issues will be considered part of conflict resolution. Conflicts are characterized by issues to enable profitable resolutions.

Like correspondence identification, there are three basic methods of issue formation. First, the linkage from feature conflicts to issues can be given. Second, derivation records can be created during specification linking components to issues; conflicts are determined by tracing from conflicting features to their common issues[46]. Finally, heuristics can use feature conflicts with constraints derived from the context to recognize

[7]Specifications and perspectives can be reused by retrieval and integration. See[46] for a simple example.

issues. This is essentially plan recognition. It is complicated by multiple agents and their interacting plans. The recognizer must map the conflicts to a common set of goals for negotiation.

5.4. Domain Goal Knowledge

A conflict characterized as a choice between mutually exclusive alternatives is unlikely to be resolved adequately. Instead, conflicts should be seen as interfering means of differing goals. In nearly all but idealogical arguments, this is true.

Next, we present knowledge structures to characterize conflicts and aid in their resolution. They are based on structures manipulated by **Oz** in its application of integrative reasoning[47]. Hence, their presentation is somewhat biased. Yet, a GI must use similar structures to characterize conflicts and reason about their resolution.

The structures consist of goals, implementation preferences, and implementation meta-preferences as presented in section 4.3. However, now we extend our model to include preference scales and goal relations. These concepts are used to combine preferences and constrain resolution search.

5.4.1. Preference Scales

Comparing alternatives amongst themselves and with the ideal requires the definition of a distance function, *d*. To reduce the effects of incomensurate scales on *d*, we scale implementations by the degree they fulfill their goal[63]. [8] For example, a bowling score of 300 fulfills PerfectBowlingScore 100%. Scaling preferences by partial goal fulfillment allows *d* to combine a variety of preferences in a measure of distance. However, partial fulfillment can be difficult to define.

For example, consider WorkingSet. Implementations LoanPeriod, MaterialLimit, and Renewal *cover* WorkingSet; they are different ways to give users access to materials. Preferences are defined on the power set of covering implementations.[9] A user might prefer WorkingSet implementations by the number of component implementation they include:[10]

(null,Renewal,MaterialLimit,LoanPeriod,[Renewal,MaterialLimit],[MaterialLimit,LoanPeriod]
[Renewal,LoanPeriod],[Renewal,MaterialLimit,LoanPeriod])

Clearly, [Renewal,MaterialLimit,LoanPeriod] is 100% and null is 0%. However, values between can be scaled a variety of ways. Now, users define both preferences and scaling.

5.4.2. Goal Relations

Scaling provides a distance measure, which in turn defines the ideal, anti-ideal, and the nondominated set. The dimensions of the scales describe the broad bounds of resolution search. These bounds may be constrainted through goal relationships. Such relations also provide opportunities of conflict dissolution and compensation (§ 5.5.2).

[8]For example, changing a scale from meters to kilometers can alter the perception of the ideal.

[9]We list component implementations while implicitly recognizing they may be combined in the support a goal.

[10]Preferences are enclosed in parenthesis with better values to the right: (1,2,3,...). When a value consists of multiple implementations, they are enclosed in brackets: ([1,2],[3,4],...). Equivalence sets (no preference) are enclosed in braces: (1,{2,2.0},3,...).

Besides the implicit support relation between implementations and goals, **Oz** uses four other relations: *inherent conflict*, *obstruction*, *partial obstruction*, and *linear correlation*. A GI would use these along with more expressive relations.

(1) *Inherent Conflict*
Causality determines if goals *inherently conflict*. For example, definitional opposites: Hot $\longleftarrow\!\neq\!\longrightarrow$ Cold, Light $\longleftarrow\!\neq\!\longrightarrow$ Dark.

(2) *Obstruction*
When all implementations of goal A inherently conflict with all implementations of goal B, then A and B *obstruct* each other. Obstruction propagates inherent conflict up the goal graph.

(3) *Partial Obstruction*
When some implementations of goal A inherently conflict with some implementations of goal B, then A and B *partially obstruct* each other. Partial obstruction also propagates inherent conflict up the goal graph.

(4) *Linear Correlation*
Linear correlation is a very specific type of relation. It demonstrates how detailed goal relations can be represented. Goal A is *linear* to goal B if both have only one preference scale and the implementations on the two scales exhibit a linear relationship; A *linearly supports* B when the relation is positive, A *linearly obstructs* B when the relation is negative. For example, EmployeeHours can linearly support EmployeeDollarCosts.

Relations based on inherent conflict or causal correlation (e.g., linearity) divide the preference space into feasible and infeasible implementations. "Ideal" resolution search only picks feasible implementations, while using infeasibles to rank them.

For some domains, many casual relations are known. For them, more specific relations can be defined. For example, *dependent conflict*: two implementations not inherently conflicting (e.g., [Gasoline,Oxygen]) but, combined with another, they obstruct a goal (e.g., [Gasoline,Oxygen,Heat] $\longrightarrow^{*}$ Safety).

The more relations we can express about a domain, the better our model. However, we must be careful about expressing noncausal conflicts as they constrain the initial search dimensions. Also, our resolutions methods are not directly applied to inherent conflicts. So, when A $\longleftarrow\!\neq\!\longrightarrow$ B is based on intuition, an integrative resolution may exist, but will not be sought.

Causal relations aid in detecting goal interactions. Yet, even precise mathematical relations can be inadequate. We cannot model domains such that we can predict all interactions of goal implementations[55, 63]. Except in simple domains, we can always invent new implementations having new interactions[26]. Thus, we analyze prototypes to detect goal interactions.[11]

When goals are found to interfere during integration, we search for alternative goal implementations. One could actually build and compare specifications; however, that would be very inefficient. We would rather search through preference space which abstracts actual specification interactions. However, this search is only as precise as the causal relations linking domain goals.

When a satisfactory resolution cannot be found, **Oz** applies compensation or dissolution methods. Now, only the partial obstruction relation is used to find compensation. If a user can't get satisfaction of goal A, goals that partially obstruct A are considered for negotiation (see § 5.5.2).

[11]This does not preclude a recognition procedure which caches such analysis.

Satisfaction is assessed with perspectives. Users needs are expressed through domain goals and relations. A perspective is a "copy" of the domain model marked with user preferences. Preference satisfaction can be determined by tracing the links connecting specification components to goals (§ 5.3). Good specifications achieve near the maximum for most preferences. Perspectives represent user requirements; they evaluate specifications; they allow for reasoning about alternate means when requirements are not met. That reasoning process is presented next.

5.5. Conflict Resolution

A feature conflict appears when users support interfering specifications. Users can cooperatively explore a space of specifications constrained by the environment and their understanding of it. Opportunity for conflict resolution stems from (1) differences in preferences, and (2) misunderstandings of the domain.

Conflict resolution aims to dissolve conflicts, and if that fails, to find "fair" compromises. In both cases, negotiators use two basic insights.

- *Separation of Means and Ends*
  Negotiators recognize that most conflicts concern *means*, i.e., methods by which goals are pursued. Separation of means from *ends*, the desired end states, allows one to accept substitute means to achieve one's goals; sometimes these alternate means are superior to the original.
- *Search for Alternatives*
  Once a conflict situation is understood to concern n different goals, one can effectively search this n-dimensional space for compromise alternatives. Furthermore, knowledge of the dimensions (goals) and their interactions aids the formation of novel alternatives.

These two insights form the gist of *integrative reasoning.* Resolution space was introduced in section 4.3, while separation of means from ends is captured in *Issue Formation* (§ 5.3) and implementation preferences (§ 4.4.1).

The following subsection will introduce an analytic method which derives resolutions in preference space. The multicriteria simplex method, as employed by **Oz**, uses only linear constraints and simple preferences. After its presentation, we introduce methods which remove, reduce, or compensate conflicts.

5.5.1. Analytic Methods

While heuristic methods are mainly concerned with altering the negotiation situation, analytic methods try to make the best of the given situation. One method is simply to enumerate all the possible choices and let an arbitrator pick the "best" compromise. However, such a strategy is ineffective if the choice set is large. It is more practical to generate nondominated alternatives. This set is usually smaller the the initial set, thereby simplifying search. However, both the initial set of alternatives and the nondominated set, N, may be infinite.

- *Multicriteria Analysis*
  In multicriteria analysis, agents scale their implementation preferences. Then, the method determines the ideal, anti-ideal, and nondominated set. Zeleny's multicriteria simplex method (MCSM) is a modification of the simplex linear programming method. It allows for the simultaneous maximization of several preference functions, cf. [63].

MCSM is efficient at generating alternatives. However, it only handles linear goal relations.[12] Also, users must agree on a single implementation preference and its scaling for each goal; they can only disagree on the direction of improvement.

Utility analysis is not limited by simple preferences or linear relations. However, the model has other limitations (§ 4.3.3). Moreover, the multicriteria model is not inherently limited. Unfortunately, analysis tools have yet to support its generality.

Next, we consider methods which compensate or dissolve conflicts. By doing so, they can move the nondominated set closer to the ideal.

5.5.2. Heuristic Methods

Goals should not be considered to be in conflict until so proven. This decision theoretic idea supports the notion that one can't sufficiently model an environment to predict the interaction of unimplemented ideas[55, 63]. Except in the most constrained worlds, one can invent new alternatives. Using this idea, we are developing *Dissolution Heuristics.*

Unfortunately, some conflicts can't be dissolved. But, compensation can be given to those interests not met by an alternative. Our *Search for Compensation* addresses this problem.

Finally, *Case-Based Modification* attempts to resolve conflicts by analogy. Here, conflicting situations are recognized and matched with previously resolved disputes. Each of these topics are presented next.

- *Dissolution*

  Consider again the conflict over the loan period (§ 4.3) It can be dissolved by the applying an appropriate heuristic. One could subdivide the inputs of the loan process and distribute the loan periods over subclasses of inputs, i.e., based on patron and book subtypes. Our *Predicate Input* heuristic suggests just this type of resolution. It is not specific to the loan goal, rather it uses the I/O behavior described in a specification language to construct a resolution. Such heuristics can be considered conflict resolving methods as in[29, 57] or patches as in[54, 58]. However, they are most closely related to the broad categories of dissolution techniques described in[42, 60]

  Note how this type of heuristic will lead us to consider other goals: resource and patron subtypes. Moreover, their inclusion can be recorded with a specific justification for their existence, i.e., reconciliation of multiple loan periods. This type of conflict-driven resolution has been observed in design. "Resolving the conflicts among system requirements created a feedback cycle in which many groups provided inputs or constraints that had [to] be negotiated into a design."–p. 1277[8].

- *Compensation*

  Compensation also increases the number of goals under consideration. Again from the example, instead of dealing with the one goal of loan period, we attempt to bring in other goals which can be traded-off against each other, i.e., using integrative bargaining techniques such as coupling/decoupling and *quid pro quo*, cf. [42].

  Pruitt has defined three basic types of compensation: specific, homologous, and substitute. These run the gamut from compensation directed to the specific needs blocked in a conflict to general compensation unrelated to the original needs expressed in the conflict. Searching ancestors of conflicting goals is one way to derive compensations in order of increasing generality. This is a good strategy, since the more general substitute compensations tend to be more difficult to accept and

[12]Nonlinear functions make analysis much more difficult; however, one could apply a cutting-plane

require greater satisfaction than direct compensation[42].

With the example, one could maintain the loan period, but in compensation do some combination of increasing material limit and renewals; also, using the partially obstructs relation, recall can be reduced. These goals are found by examining the goal graph (figure 3) to find siblings of the conflicting goals. However, ultimately it is the evaluation of whole perspectives that we want to maximize. Hence, increasing local goal satisfaction is a heuristic which gives way to searching up the goal graph in case of an impasse. For example, if a satisfactory adjustment of WorkingSet could not be obtained, one might consider compensation for ancestral goals. For example, reducing user Fees or Fines would increase the evaluation of the ancestor ResourceUsage.

- *Case-based Modification*

  Sycara has demonstrated the usefulness of case-base modification in labor relations–a domain where precedent based reasoning is partially dictated by the law[56]. Using records of previous disputes which include all impasses and resolutions, **PURSUADER** proposes resolutions based on modifications of analogous cases. Modifications take into account any significant differences, e.g., inflation, location, market-share.

  Both cases that 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 compromises, dissolutions, and compensations from consideration. The case-base is a cache for integration knowledge, as well as a store of unenunciated negotiation knowledge.

Even after an alternative has been chosen, the job is not done. A resolution must be implemented, as described next.

5.6. Implementing a Resolution

Implementing a resolution amounts to remapping goals to specification components. This process is the inverse of issue formation and has three basic methods. First, the

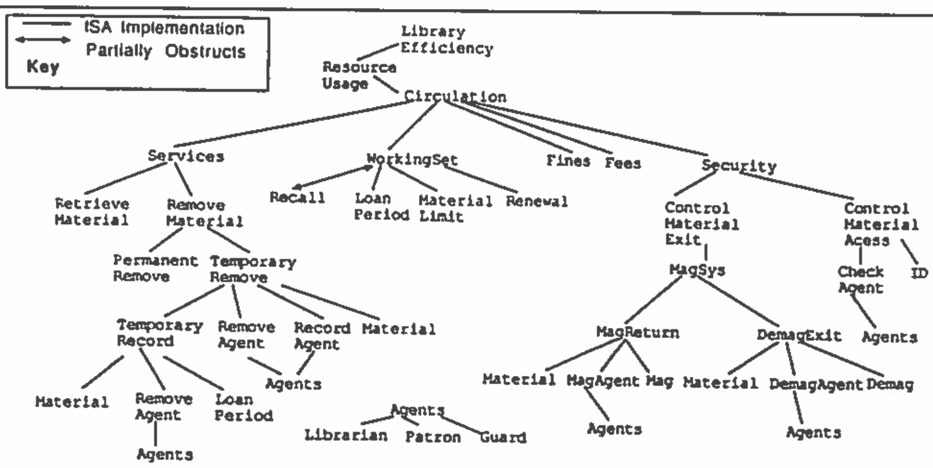

Figure 3. Part of a Library Goal Relations Graph.

linearization as an approximation. cf. appendix C in[24].

mapping from a resolution to a specification can be given. However, a direct plan instantiation procedure is unlikely because of the creation of novel resolutions.

Second, a specification can be created by a design agent. This process will be guided by the resolved goals. If conflicts arise during this process, they can be resolved using the integration paradigm. However, such conflicts are less likely to occur because many interactions have already been considered during negotiation.

Finally, the original specifications can be patched. Noninterfering components are kept. Interfering components are modified according to goals derived from negotiation. Both Robinson[45] and Feather[15, 17] have employed a variation of this process, whereby a single specification is modified to contain all user goals.

5.7. Evaluation

The integration paradigm seems well suited to aid requirements negotiation. Through its use of integrative reasoning, it addresses negotiation freedoms and their implications. However, its full generality has yet to be explored. Particularly lacking are: (1) a substantial goal relations language, (2) preference and scaling languages, (3) a general multi-criteria search method, and (4) substantial dissolution and compensation knowledge. Yet, even a severely constrained integrator can be useful. The next section demonstrates this using **Oz**.

6. An Automated Example

In this section we illustrate MPSD with an example. We focus on conflict resolution by integrating three very simple specifications. The example is simplistic for tutorial clarity and because of tool simplicity.

Oz is an experimental MPSD assistant. Now, it manages the specification process and aids some procedures interactively. Unfortunately, automation of conflict resolution is severly limited. This example shows current and planned automation.

Figure 4 shows three prototypes of a library borrow operation. They represent (simplistic) ideal specifications from the perspectives of a patron, a librarian, and a security officer.[13]

The patron's perspective supports material removal from a library. The librarian supports this goal, but only if records are kept. The security officer is interested in preventing theft. His prototype supports this goal by magnetic scanning.

Now that our users' goals have been prototyped in specifications, we begin the five step process of integration.

6.1. Correspondence Identification

Correspondence identification determines equivalences like $\text{Exit}_{\textbf{Patron}} \equiv \text{Exit}_{\textbf{Librarain}}$. The analyst decides which specification components represent the same concept. We support this process with automated comparison guided by predicates. The analyst controls predicate application by marking them. He can use NamesCorrespond, TypesCorrespond, and define his own predicates (in Lisp). Once initial correspondences are created, they can be edited.

[13]The specification language is an extension of Petri nets[41]; bubbles are conditions on procedures (boxes).

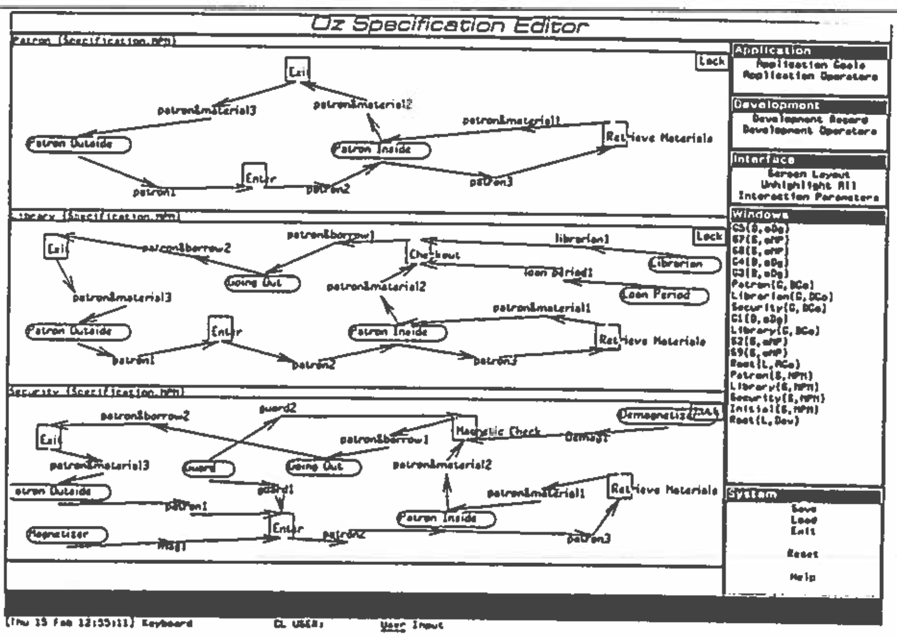

Figure 4. Three Specification Prototypes of RemoveMaterial.

6.2. Conflict Detection

After correspondences are determined, conflicts are detected. In our specification language, components are defined by a type and its slots. For example, Exit is a *transition* with slots *inputs* and *outputs* (among others). A feature conflict is noted between $\text{Exit}_{\textbf{patron}}$ and $\text{Exit}_{\textbf{librarian}}$ by a different *place* in the input slot. In this way, differences between correspondences are automatically detected.

6.3. Issue Formation

Next, *direct issues* are identified by tracing *incorporation links* from conflicting specification components to goals (figure 5). These links are created when perspectives are mapped to specifications.[14] Differences between **Patron**, **Librarian**, and **Security** are traced to the following goals (see figure 3):

> RemoveMaterial, TempMaterialRecord, LoanPeriod, RecordAgent, ControlMaterialExit, ControlAcess, CheckAgent, MagMonitorSys, MagReturn, DeMagExit, MagAgent, DeMagAgent.

User preferences associated with these goals form a resolution space. However, this 12-dimension preference space can be broken into a hierarchy of subspaces. For example, observe that Security is the root of a hierarchy of goals (ControlMaterialExit, ControlAcess,

[14]Besides providing negotiation support, these links are used to track the status of goals; goals may, or may not be *incorporated* into a specification. Furthermore, one may explain the rationale for system features based on the negotiations which created them. For example, one could explain that the admission system's card file (of § 2) was provided to add variety to U's workday and to provide an index backup. If U were replaced and another backup mechanism available, negotiations (design) could be profitably reopened.

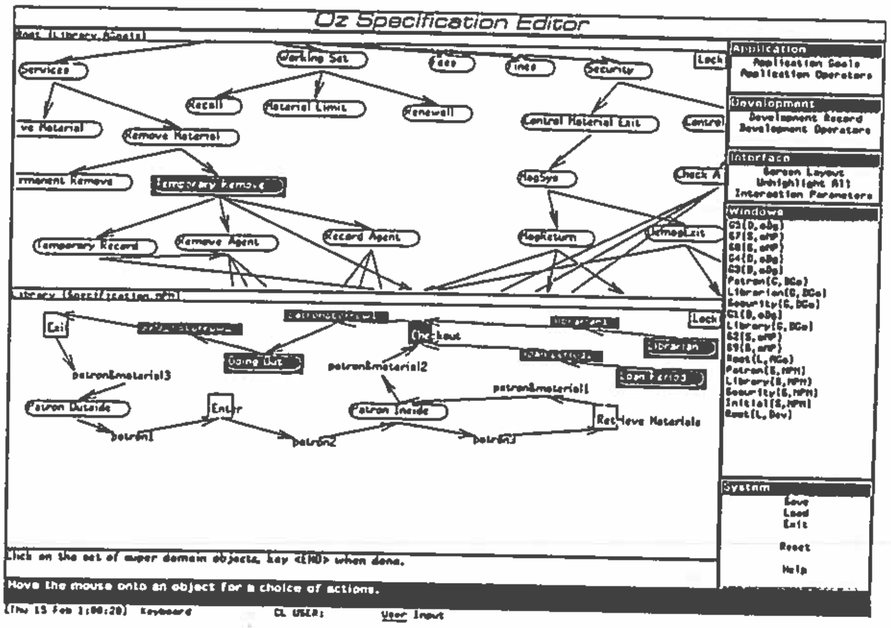

Figure 5. An **Oz** Depiction of Incorporation Links.

MagMonitorSys, ...). When the Security conflict is resolved, it will effect lower level preferences. Using this idea, we form an *issue tree.*

Figure 6 shows the issue tree for the direct issues. It's simply the domain model linkage between goals. This tree provides control over the resolution process. Issues are resolved from the root(s) to the leaves. This strategy provides abstraction over a (possibly very large) set of issues. Also, it resolves issues in order of importance and scope. Next, we show how conflicts are resolved using this tree.

6.4. Conflict Resolution

Five steps are used to resolve the 12 issues. Step one is shown nearly in its entirety. The five steps that follow are summarized.

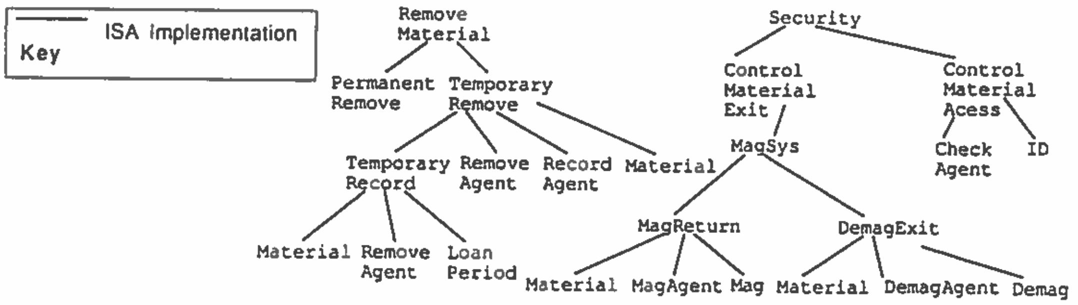

Figure 6. Resolution Issue Tree.

6.4.1. Loaning and Security

Conflicts associated with RemoveMaterial and Security are resolved first. These two goals are the roots of the issue trees. Table 1 shows these two issues, their scales, and user perspectives. Because we are employing Zeleny's multi-criteria simplex method to find the ideal, anti-ideal, and nondominated resolutions, implementations are only scaled once. This limitation forces all users to agree on the degree implementations satisfy their goals. For example, all agree that PermantRemove achieves RemoveMaterial at 90%. However, users can express different directions of preference. They can have no preference (i.e., ---), or prefer to maximize (i.e., Max) or minimize (i.e., Min) achievement along a scale. Table 1 reflects **Patron**'s desire to minimize security, and **Librarian**'s and **Security**'s desires to maximize it.

Issues			
Scale	*RemoveMaterial*	*Security*	
90%	PermantRemove		
40%		[CME,CMA]	
30%	TemporaryRemove	CME	
10%		CMA	
0%	None	None	
Perspectives			
Patron	Max	Min	
Librarian	Min	Max	
Security	---	Max	
Goal Relations			Limit
	1	1	<= 70

Table 1: Issues for compromise c_1.

Table 1 also illustrates the use of a ***linear obstructs*** relation. Under each issue is a constant multipler. Combined with the limit, they form a constraint on the combined implementations of RemoveMaterial and Security: (1 * RemoveMaterial) + (1 * Security) $\leq$ 70%. The constraint is used to limit the combined cost of these systems. Clearly, this is a poor representation. Mappings from percent achievement to dollars are implicit in this constraint. Such mappings, as well as multiple scales, will be part of a future resolution search method.

Our current search method is Zeleny's MCSM. Using it, scales form search dimensions, goal relations are constraints, and perspectives are functions to be maximized or minimized. MSCM first derives the nondominated extreme points, N_{ex} = {[0,0],[70,0],[0,70]}, as determined by the goal relations. The nondominated set potentially contains these and other points: $N = N_{ex} \cup$ {[35,35],[30,40],[40,30] ...}. Using IDEA, the analyst weight the various implementation preferences, identifies trade-offs, and picks a compromise. The pair [30,40] is the chosen scale-space compromise. Mapped to goal implementations, it corresponds to:

c_1=[TemporaryRemove,[ControlMaterialExit,ControlAcess]].

Compromise c_1 represents a library system which uses a temporary removal mechanism (TemporaryRemove) to implement material removal. Security is obtained through two abstract mechanisms which control the exit of material and access of patrons.

Compromise c_1 represents the resolution of two conflicting goals between three agents. The analyst moved from the abstract issues of c_0=[RemoveMaterial,Security] to the more refined concerns of c_1. Similarly, the next four compromises will negotiate issues until leafs of the issue tree are derived.

MSCM is fully automated, while IDEA is still a simple weighting aid. Open areas of research include: (1) expanding MCSM to include a general preference reasoner to aid search and (2) expanding IDEA to include a general arbitration system which can reason about agents and derive meta-preferences.

6.4.2. Loan, Exit, and Access

Continuing with the example, three new issues must be resolved: TemporaryRemove, ControlMaterialExit, and ControlAcess. Their resolution follows the pattern of the first compromise. Compromise c_2 resolves temporary removal with a recorded transaction; material exit is limited with a magnetic monitoring system (MMS); finally, access is controlled with an identification system.

c_2=[[TempMaterialRecord,RecordAgent],[MMS,[ID,CheckAgent]]]

6.4.3. Loan Records, Agents, and Monitoring

Table 2 shows the issues involved in the next resolution. Although ID was an issue in c_2, it is not part of c_3 since it is a primitive (leaf) goal.

Issues					
Scale	*TempMatRecord*	*RecordAgent*	*MagMonitorSys*	*CheckAgent*	
100%	Complete	Video	[MagReturn,DemagExit]	[Video,Guard]	
30%	[Mat,Agent,LP]	Librarian		Guard	
20%	[Mat,Agent]	Guard		Librarian	
10%	Material	User			
Perspectives					
Patron	Min	Min	Min	Min	
Librarian	Max	Max	---	---	
Security	---	Min	---	Min	
Goal Relations					Limit
		1		1	Min

Table 2: Issues for compromises c_3 and c_4.

The derivation of compromise c_3 is like that of the previous two. It determines the loan record and recording agent, the magnetic system components, and the identification checking agent.

c_3=[[[Material,RmvAgent,LoanPeriod],Librarian],[MagReturn,DeMagExit],Guard]

While this is the best compromise in *N*, it is still unsatisfactory according to the analyst.[15] He consults his dissolution rules and applies the *Functional Sharing* heuristic to Librarian and Guard.

> **Functional Sharing**
> If (1) processes *a* and *b* have the same inputs, except for sets *a.i* and *b.i.* (2) processes *a* and *b* have the same outputs except for sets *a.o* and *b.o.* (3) **.i* and/or **.o* may be null.
>
> Then create *c* with combined inputs **.i* and combined outputs **.o*, and use *c* in place of *a* and *b*. (*c* may form a limited replacement but reducing sets **.i* and **.o*).

Application of *Function Sharing* creates a new implementation, named a *WatchfulLibrarian*. It is added to the implementations of CheckAgent, RecordAgent, and all other agent goals. Finally, a new compromise c_4, closer to the ideal, is obtained.

c_4=[[[Material,RmvAgent,LoanPeriod],WatchfulLibrarian$_1$],[MagReturn,DeMagExit], WatchfulLibrarian$_1$]

We have a catalogue of rules similar to *Function Sharing*. Future research includes: (1) refining current rules, (2) extending the rule catalouge, and (3) controlling automatic rule application.

6.4.4. Loan Period, Return, and Exit

Compromise c_5 is derived in similar fashion; it is the best of *N*, but still unsatisfactory. Table 3 shows the issues involved. Goals Material, RmvAgent, and WatchfulLibrarian are left out, as they are primitive (at his point). Compromise c_5 represents a four week loan period and the components of the magnetizing and demagnetizing systems.[16]

c_5=[4w,[MagAgent,Material,Mag],[DmAgent,Material,DeMag,MagCheck]]

To better it, the analyst applies dissolution rule *Predicate Input* to LoanPeriod. Analysis of the specification inputs to the loan process reveal that both Patron and Material can be divided into subtypes. This results in new subtypes for Material: {Book,Reserve,Periodical,...} and RmvAgent: {Grad,UnGrad,Faculty,Staff,...}.

[15]In simplifying this example, we removed the Administration perspective. It prefers to minimize the number of system agents. This is illustrated in table 2 as the minimization on RecordAgent and CheckAgent. In this example, the analyst reflects the implicit dissatisfaction the Administration feels. Such dissatisfaction of preferences guides the application of dissolution rules.

[16]A four week loan period was derived from Librarian's anchored preference. An *anchored preference* defines preferences relative to a particular implementation. The Librarian prefers to minimize the distance from a two week implementation. Anchored preferences are illustrated in tables 3 and 4.

Issues			
Scale	*LoanPeriod*	*MagReturn*	*DeMagExit*
100%	∞ days		
80%		[MagAgent,Material]	[DemagAgent,Material]
20%		Material	Material
0%	0 days		
Perspectives			
Patron	Max	Min	Min
Librarian	2w	---	---
Security	---	Max	---

Table 3: Issues for compromises c_5 and c_6.

Compromise c_6 is based on the new subtypes. Now, multiple loan periods will be given on the basis of material and patron subtypes. However, from c_5 it is only apparent how Material subtypes can be incorporated:

c_6=[[2w,6m],[MagAgent,[Book,Reserve,Periodical],Mag],[DmAgent,[Book,Reserve,Periodical],DeMag]]

But c_6 is only part of the complete compromise tuple:

$c_{6'}$=[[Material,RmvAgent,[2w,6m]],WatchfulLibrarian$_1$,[MagAgent,[Book,Reserve,Periodical],Mag],
[DmAgent,[Book,Reserve,Periodical],DeMag],WatchfulLibrarian$_1$]

From $c_{6'}$ we can see how the implementations off both Material and RmvAgent can be propagated:

$c_{6''}$=[[[Book,Reserve,Periodical],[Grad,UnGrad,Faculty,Staff],[2w,6m]],WatchfulLibrarian$_1$,
[MagAgent,[Book,Reserve,Periodical],Mag],[DmAgent,[Book,Reserve,Periodical],DeMag],
WatchfulLibrarian$_1$]

However, even the complete tuple does not show the predication of LoanPeriod based on subtypes. It's up to resolution implementation to incorporate such operational descriptions into a specification. Yet, it must have some description. The resolution derivation history provides help. It contains rule applications which indirectly point to operationality. Using it, more complete operational descriptions are constructed during resolution implementation.

6.4.5. Monitoring Agents

In the last resolution, all remaining issues are primitive (e.g., Mag, DeMag, and MagCheck) save the magnetizing and demagnetizing agents. Again, the best weighted compromise is unsatisfactory.

c_7=[WatchfulLibrarian$_2$,WatchfulLibrarian$_2$]

While this compromise makes use of the WatchfulLibrarian agent type created in compromise c_4, there are still too many agents associated with the library. Compromise c_7 is only part of the complete tuple:

$c_{7'}$=[[[Book,Reserve,Periodical],[Grad,UnGrad,Faculty,Staff],[2w,6m]],WatchfulLibrarian$_1$,
[WatchfulLibrarian$_2$,[Book,Reserve,Periodical],Mag],
[WatchfulLibrarian$_2$,[Book,Reserve,Periodical],DeMag],WatchfulLibrarian$_1$]

Next, the analyst applies *Functional Sharing* to WatchfulLibrarian$_1$ and WatchfulLibrarian$_2$:

c_8=[[[Book,Reserve,Periodical],[Grad,UnGrad,Faculty,Staff],[2w,6m]],WatchfulLibrarian,
[WatchfulLibrarian,[Book,Reserve,Periodical],Mag],
[WatchfulLibrarian,[Book,Reserve,Periodical],DeMag],WatchfulLibrarian]

It is still unsatisfactory with regard to **Patron**'s LoanPeriod, so the analyst applies compensation heuristics. Examining the graph is figure 3, the analyst observes the two siblings of LoanPeriod are MaterialLimit and Renewal. Also, through the obstructs link between WorkingSet and Recall, Recall becomes an issue. These three issues support or obstruct the immediate ancestor of LoanPeriod. Compensation can be provided by heavily weighting supporting goals and minimally weighting obstructing goals. By these weights, the **Patron** perspective dominates the compromise choice.

Issues			
Scale	*MaterialLimit*	*Renewal*	*Recall*
100%	∞	∞	∞
0%	0	0	0
Perspectives			
Patron	Max	Max	Min
Librarian	20	Max	Min
Security	Min	Min	Min

Table 4: Issues for compromises c_9, c_{10}, and c_{11}.

After considering the perspectives in table 4, the best weighted compromise is c_9.

$c_9=[20,\infty,0]$

Happily, this resolution is satisfactory. However, if it was not, we would continue the *Search for Compensation* by considering goals which support or obstruct the ancestors of WorkingSet (e.g., Circulation, ResourceUsage). Constraining this search remains an open problem.

Now, we have finished the resolution process. At the top level of abstraction, our resolution issues are:

c_{10}=[RemoveMaterial,Security,MaterialLimit,Renewal,Recall]

Completely filled out, it is:

c_{11}=[[[Book,Reserve,Periodical],[Grad,UnGrad,Faculty,Staff],[2w,6m]],WatchfulLibrarian,
[WatchfulLibrarian,[Book,Reserve,Periodical],Mag],
[WatchfulLibrarian,[Book,Reserve,Periodical],DeMag],WatchfulLibrarian,
20,Unlimited,0]

Compromise c_{11} represents a library which loans materials for two weeks or six months based on patron and material subtypes. There is a 20 item checkout limit, unlimited renewal, and no recall. A single watchful librarian runs the security and checkout systems.

The above example has shown the (1) combination of analytic compromise with heuristic improvement, (2) resolution control through issue abstractions, (3) incorporation of new issues, and (4) reuse of prior issue compromises. This has been accomplished through integrative reasoning, a three step sequential process of: (1) compromise, (2) dissolution, and (3) compensation. Generalizing the control to consist of the intertwinning of these processes remains an open problem.

6.5. Resolution Implementation

Once a resolution is chosen, it must be implemented. Given our simple example, an analyst can directly create a specification. However, more typically the resolution is a small part of the overall specification. The analyst must patch the specification to include the changes implied by the resolution. Specification construction is guided by both the resolution tuple and the resolution derivation history. This remains a difficult task.

6.6. Evaluation

Even with simple examples and limited automation, we have obtained encouraging results. Integrative reasoning is a viable method of automation, and MPSD is a viable method of requirements negotiation.

Many benefits are derived from the MPSD model. It addresses: (1) conflict and preference management, (2) multiple analysts, (3) specification and compromise reuse, and (4) specification rationale. However, it rests on the assumptions that: (1) parts of a specification are better designed in isolation, and (2) specification integration adequately addresses conflicts. Our arguments throughout, as well as those for modular languages, support these assumptions. However, what about optimality?

MPSD considers how well implementations satisfy goals. Compromise, dissolution, and compensation methods are directed to local interactions. More global concerns are only satisfied through local interactions. Overall optimization is not directly addressed. Instead, multiple optimizations are considered.

If a single goal were to be satisfied, then overall utility could be defined. This sole arbitrator would resolve all "conflicts" in its favor; in fact, conflict cannot exist without goal competition. Conflicts occur when multiple goals interfere. Thus, the idea of universal optimization is foreign to a model which allows multiple goals of (nearly) equal importance.

7. Conclusions

In this paper, we have presented Multiple Perspective Specification Design (MPSD) and *integrative reasoning*. This model of requirements and specification acquisition, and its automation, support three basic freedoms: freedom of preference, conflict, and compromise. By supporting these freedoms, MPSD directly addresses the intertwining of requirements acquisition and specification design.

Swartout and Balzer have noted how some specification changes are driven by an understanding of the implementation[55]. Of course, other changes are driven by changing user requirements. Many research projects are concerned with the cyclic process of specifying, implementing, and modifying a specification. "It is only because we have allowed this development process to occur, unobserved and unrecorded, in people's heads that the multi-step nature of this process was not more apparent earlier."[55]

While the intertwining process is recognized, it is not directly supported; most tools simply speed up the process (e.g., rapid prototyping) without tying the observed conflicts into a system which aids their resolution and respecification. MPSD and its surrounding support can be seen in this light. It aids in the description of requirements, their specification, recognition of conflicts, resolution, and their respecification.

This multi-step process is similar to specification implementation. Until recently, this development process has gone on unobserved and unrecorded. Requirements acquisition, like specification implementation, is an intertwined process. In both processes, the common thread is the recognition and resolution of goal conflicts. The automated support of negotiation freedoms directly address intertwined processes.

ACKNOWLEDGEMENT

Thanks to John S. Anderson for his comments on previous drafts of this paper. Also, thanks to Arthur Farley for his insightful critique of the goal terminology. Robert Helm provided endless examples (and counter examples) to consider. Also, thanks to Martin Feather for his parallel elaboration model and his critiques of this project. Finally, we thank the National Science Foundation for their support via grant CCR-8804085.

REFERENCES

1. E.W. Adams and R. Fagot, "A model of riskless choice," in: Eds. W. Edwards, A. Tversky, *Decision making*, (1967) 284-289.
2. K.J. Arrow, "Public and private values," in: Eds. S. Hook, *Human values and economic policy*, New York University Press (1967) 3-31.
3. R. Balzer and N. Goldman, "Principles of good software specification and their implications for Specification Languages," *Proceedings of IEEE Conference of Specifications of Reliable Software*, (1979) 58-67.
4. 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.
5. S. Bendifallah and W. Scacchi, "Work structures and shifts: an emperical analysis of software specification teamwork," *11th International conference on software engineering*, (May 1989) To appear.
6. A.H. Bond and L. Gasser, *Readings in distributed artificial intelligence*, Morgan Kaufmann, San Meteo, California (1988).
7. 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.
8. B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *CACM* **31** (November 1988) 1268-1287.
9. Randall Davis and Reid G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence* **20** (1983) 63-109.
10. M. Deutsch, *The resolution of conflict: constructive and destructive processes*, Yale University, New Haven (1973).
11. T. Dietterich and D. Ulman, "Artificial intelligence approaches to design," *Artificial Intelligence in the Northwest*, (October 22-24 1985) 8/3.
12. Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Transactions on Computers* **C36** (1987) 1275-1291.
13. E.H. Durfee and V.R. Lesser, "Using partial global plans to coordinate distributed problem solvers," *Transactions on computers* **C-36** (1987) 1275-1291.

14. T.G. Evans, "A program for the solution of a class of geometric analogy intelligence test questions," in: Eds. M. Minsky, *Semantic information processing*, MIT Press , Cambridge, Mass (1968) 271-353.

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

16. M.S. Feather, "Language support for the specification and development of composite systems," *Transactions on Programming Languages and Systems* **9** (April 1987) 198-234.

17. M. S. Feather, "Constructing specifications by combining parallel elaborations," *Transactions on Software Engineering* **15** (February 1989) To appear (Also available as Technical Report RS-88-216 from ISI).

18. L. Festinger, *Conflict, Decision, and Dissonance*, Tavistock Publications, Ltd., London (1964).

19. S. Fickas, S. Collins, and S. Olivier, "Problem acquisition in software analysis: a preliminary study," CIS-TR-87-04, University of Oregon (January 1988).

20. A. Finkelstein and H. Fuks, "Multi-party specification," *5th International workshop on software specification and design*, (1989) 185-195.

21. M.R. Garey and D.S. Johnson, *Computer and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, New York (1979).

22. Dedre Gentner, "The mechanisms of analogical reasoning," in: Eds. S. vosniadou, A. Ortony, *Similarity and analogical reasoning*, Cambridge University Press (1989) 199-242.

23. M.P. Georgeff, "A theory of action for multiagent planning," in: *Proceedings of 1984 conference of the AAAI*, Morgan Kaufmann Publishers (1984) 121-125.

24. A. Goicoechea, D.R. Hansen, and L. Duckstein, *Multiobjective decision analysis with engineering and business applications*, John Wiley and Sons (1982).

25. Carl Hewitt, "Offices are open systems," *Trans. on Office Information Systems* **4** (1986) 271-287.

26. E. Jantsch, *Technological Forecasting in Perspective*, Organization for Economic Co-operation and Development, Paris (1967).

27. S. Kedar-Cabelli, "Purpose-directed analogy," in: *Proceedings of the International Conference of the Cognitive Society*, (1985) 150-159.

28. R.L. Keeney and H. Raiffa, *Decisions with multiple objectives*, John Wiley and Sons, New York (1976).

29. M. Klein and S. C-Y Lu, "Run-time conflict resolution in cooperative design," *AI and Design Workshop*, (1988) To appear.

30. D.F. Kohl, *Circulation, interlibrary loan, patron use, and collection maintenance: A handbook for library management*, ABC-Clio Inc. (1986).

31. W.A. Kornfeld, "The scientific community metaphor," *IEEE Transactions on Systems Man Cybern* **SMC-11** (January 1981) 24-33.

32. Susan Lander and Victor Lesser, "Negotiation among cooperating experts," *AI and Design Workshop*, (1988) To appear.

33. D.B. Lenat, "The nature of heuristics," *Artificial Intelligence* **19** (1982) 189-249.

34. D.B. Lenat, "The nature of heuristics II," *Artificial Intelligence* **21** (1983) 31-59.

35. D.B. Lenat, "The nature of heuristics III," *Artificial Intelligence* **21** (1983) 61-98.

36. Philip E. London and Martin S. Feather, "Implementing specification freedoms," *Science of Computer Programming* **2** (1982) 91-131.

37. Marc Luria, "Goal conflict concerns," *IJCAI-87*, (1987) 1025-1031.

38. Jack Mostow, "A problem-solver for making advice operational," *AAAI-83*, (1983) 279-283.

39. J. Mostow, "Towards better models of the design process," *AI Magazine* **6** (Spring 1985) 44-57.

40. J. Mostow, "A preliminary report on DIOGENES: progress towards semi-automatic design of specialized heuristic search algorithms," ML-TR-27, Department of Computer Science, Rutgers University (October 1988).

41. J. L. Peterson, "Petri nets," *Computing Surveys* **9** (September 1977) 223-252.