

A Mixed Quantitative/Qualitative Method
for
Evaluating Compromise Solutions
to
Conflicts in Collaborative Design

Dennis Bahler

Dept. of Computer Science
North Carolina State University
Raleigh, NC 27695-8206 USA

Catherine Dupont

Bell Northern Research
Research Triangle Park, NC 27709 USA

James Bowen

Dept. of Computer Science
National University of Ireland
Cork, Ireland

Abstract

Conflicts are likely to arise among participants in a collaborative design process as the inevitable outgrowth of the differing perspectives and viewpoints involved. The opportunities for conflict are magnified if many perspectives are brought to bear on a common artifact early in the design process, as in concurrent engineering or integrated engineering. Design advice tools can assist in the process of resolving these conflicts by making critiques and suggestions conveniently available to design participants, and by offering a fair means of evaluating and comparing suggested alternatives for compromise solution. In previous work [Bahler *et al.* 1994a, Bahler *et al.* 1994b] we introduced a protocol based on notions of economic utility by which design advice systems can recognize conflict and mediate negotiation fairly. This protocol allowed design teams to express the desire to maximize or minimize the values of design parameters over totally-ordered bounded domains of values, such as real numeric intervals. In this paper we extend this approach by allowing expressed preferences of design teams to be qualitative as well as quantitative, by allowing teams to express interest in parameters before they actually come into existence, and by relaxing many other of the earlier restrictions on the ways teams may express their preferences.

1 Introduction

Concurrent engineering is an approach to design in which, in order to avoid costly redesign, aspects of the product life cycle such as manufacturability, testability, repairability, and marketing are taken into account as early in the design process as possible. Because sufficient expertise to accomplish these goals rarely resides in individuals or even small groups, concurrent engineering requires cooperation among many experts from various disciplines [Bowen & Bahler 1991b]. The goal of concurrent engineering is to produce better designs by sharing these viewpoints and critiques in timely fashion. Since the information management needs of concurrent engineering are extensive, advice tools become necessary to bring the various viewpoints and critiques to bear. Even though all these different clients are members of a common design team, their decisions may conflict with one another and it may not be straightforward to find a compromise. Because of this, an essential feature of design advice tools is some form of support for negotiating compromise solutions to such conflicts. In this paper we expand upon a previously-described negotiation protocol we have devised for use in such design advice tools.

2 Conflict in collaborative design

It is a wise assumption that in a collaborative design effort each participant's viewpoint consists partly of various self-serving concerns which contribute to his evaluation of the design. These concerns may be put at risk by the other participants, so each member of a design team is from time to time called upon to defend his interests, treating the others as potential adversaries. Even the most accommodating members of a design team inevitably act from time to time in a competitive fashion by virtue of their background in disparate disciplines; this gives rise to different perspectives and may lead to attempts to alter the proposed design aimed at improving their own interests in the design process. The potential for conflict and the need for negotiation in such a context are obvious. On the other hand, in the end all agents possess a common high-level goal: to agree on the design of the best product possible.

Our ambitions in support of negotiation are somewhat different from those of many investigators who have touched on the subject. In our approach, various members of the design team, called **clients**, interact with our design advice system, making decisions which create design objects and parameters, or give or retract values for existing parameters. In the event of conflicting decisions,

compromise proposals are elicited by our system from clients, which can be either humans or other computer-based systems, and the negotiation system provides support that encompasses the detection of potential conflicts involving several clients, and the collection and evaluation of these compromise proposals. This might be called **client-assisted negotiation**, by analogy with the more general notion of client-assisted inference [Bahler *et al.* 1993]. It is our view that partially automated negotiation, and more generally partially automated design, is a more appropriate ambition in the development of design advice systems. Client-assisted inference of this sort is imperative, we believe, for reasons of both computational complexity and organizational culture.

We make no attempt to generate client preferences or compromise proposals automatically, or to try to match each case of conflict with a previous experience. Those few discussions of client-assisted negotiation support that have appeared in the literature [Bowen & Bahler 1992, Werkman 1991, Werkman 1992] present mostly general guidelines and ad-hoc methods rather than detailed specifications or discussion of actual implementations. We began to address methods to fill this gap in [Bahler *et al.* 1994b], but that work imposed restrictions on what designers could express by way of preference and proposals for design modifications. This work takes the work one step further, by relaxing some of those earlier restrictions.

This work may be contrasted both with research seeking to develop cognitive models of human adversarial negotiation and conflict resolution [Sycara 1989] and with work seeking essentially to replace human negotiators with automated systems [Erman *et al.* 1988, Gasser *et al.* 1987, Rosenschein & Genesereth 1985].

3 Our negotiation protocol

In the context of client-assisted design-advice systems, a negotiation protocol should first of all be fair, where fairness may be defined by three criteria. First, it should treat all clients equally. Second, it should avoid any situation where a client could be greatly dissatisfied while other clients see their position only slightly improved. Finally, no client's priorities should have precedence over any other's.

This section describes a negotiation protocol offering the following capabilities:

- The detection of potential conflicts involving several clients or teams of clients;
- The collection of compromise proposals from the clients;
- The algorithmic evaluation of these proposals using a Plan Evaluator; with
- No requirement that one client be “first among equals.”

We acknowledge that fairness, while essential, may be only one of the qualities desirable in a conflict arbitration mechanism. For a scheme which seeks to equitably allocate resources among design participants by using an artificial economy based on a numeraire, see [Wellman 1994].

Our method automates the negotiation process only partially, by defining the properties that the solution must satisfy, and then employing procedures that compute agreement points that satisfy these properties. Specifically, in the protocol we describe, clients are required to define issues, which set forth their interests in the existence and values of design parameters. In the event of a conflict, our protocol first invites all involved parties to present compromise solutions; the negotiation facility is then responsible for comparing and ranking the solutions — quantitatively if possible, less precisely if necessary — and suggesting the most suitable compromise. The goal of comparing the collected compromise solutions is to select the proposal that can best satisfy all the clients collectively.

3.1 Terminology

Very broadly, our model of the collaborative design process is one in which the participants collectively seek to simultaneously satisfy a set of requirements represented as expressions in a formal language such as predicate calculus, by coming to consensus about the semantics of the symbols contained in those expressions [Bahler & Bowen 1995]. In our approach this process is extremely general and encompasses various activities, including compositional design — in which multi-perspective consensus is reached on what the set of design parameters should be — and parametric design — which involves consensus over what the values of those parameters should be.

We represent design information in the form of constraints, and interaction with our design advice system consists of adding or retracting constraints, or creating/destroying parameters over which constraints are defined. A description of the implementation language we have been using for several years to build such advice systems is given in [Bahler & Bowen 1995, Bahler *et al.* 1994a, Bowen & Bahler 1992, Bowen & Bahler 1991a].

3.2 Plans

To clarify our terminology, we begin with a few definitions. A **constraint network** is a triple $\langle U, X, C \rangle$ in which

- U is a universe of discourse,
- X is a non-empty, finite tuple of q non-recurring parameters,
- C is a non-empty, finite set of r constraints, $C_1(T_1), \dots, C_r(T_r)$, with the following characteristics:
 - each constraint $C_k(T_k) \in C$ restricts the values that may be assumed by the a_k members of T_k , a sub-tuple of X ;
 - each constraint $C_k(T_k)$ is a subset of the a_k -ary Cartesian product \mathcal{U}^{a_k} .

We define a **plan** as a tuple of new constraints over some subset of parameters of X , where $\langle U, X, C \rangle$ is a constraint network. Operationally, a plan consists of a set of suggested alterations to a constraint network. Such actions may be any of the following:

- *Adding a constraint.* Among the constraints permitted in plans are the usual algebraic equalities such as $X = 10.3$, which will assign the value 10.3 to the parameter X , or linear equations such as $8 \times X - 2 = Y$ where Y has a known value, say 0, in which case the parameter X will be assigned the value 0.25. Also permitted are syntactically more complex constraints such as inequalities and quantified logical expressions, in contrast to [Bahler *et al.* 1994b].
- *Retracting a constraint.* These actions alter the network by removing existing constraints, and may have other effects as well via constraint processing. For example, assume that we have two constraints referring to a parameter X : $X < 0$ and $X + Y = 3$, and that the current value for Y is 0. There is a constraint violation since X cannot simultaneously equal 3 and be less than 0. But if a client retracts the constraint $X < 0$, X will be assigned the value 3.
- *Creation of a new parameter.* These suggested alterations cause new design parameters to come into existence for the first time. This action is the means by which we support configurational design [Bowen & Bahler 1991a, Bowen & Bahler 1993], since parameters must obviously exist before parametric design can give them values.

A parameter is affected by a plan if its existence or value assignment would be affected by the application of the plan. Of course, it is always possible that an alteration of the network may violate a constraint, either within the perspective where the alteration originated or some other. In a constraint-based system such as ours, such contingencies are handled by the regular constraint-violation mechanism, and the negotiation facility need not be called into play.

A method for selecting a compromise solution for a set of clients consists of requiring the prior declaration of preferences from all the clients and of computing the quantitative satisfaction (or dissatisfaction) caused by each candidate solution with respect to the expressed client preferences. In our context, the payoff to a client from a compromise solution is the overall utility for that client based on his declared preferences.

Our notion of client utility is essentially based on the multiattribute form of simple von Neumann/Morgenstern expected utility, a numerical measure applied to comparisons of alternatives which obeys axioms of ordering, independence, and continuity. This form of utility can be shown to be unique up to a positive affine transformation [Fishburn 1988]. This idea of utility is only one alternative found in the considerable literature in economics [Daboni *et al.* 1986, Samuelson & Nordhaus 1992], operations research [Hillier & Lieberman 1990], and decision theory [Keeney & Raiffa 1976]. We note in passing that there are intriguing generalizations of von Neuman/Morgenstern utility to nonlinear, nontransitive forms which may be more in keeping with psychological observation of human behavior [Tversky & Kahneman 1981], but further discussion of the utility-theoretical aspects of this work is beyond the scope of this paper.

3.3 Issues

Client preferences are represented by **issues**, which are expressed by design participants prior to beginning the actual design process. An issue consists of a set of parameters, together with an indication of the desired direction of change. In addition, parameters defined on bounded totally-ordered domains are qualified by a **differential utility function**, which indicates the impact on marginal utility of a given change in value. A client can have only one issue. In practice, issues are not normally shared among clients, but it is extremely common for issues to share parameters.

The contents of the issue list parameters, either individually or collectively by domain, which are involved in the issue with the desired direction of change (**min** or **max**), their respective relative weights within that issue, and, in the case of bounded totally-ordered domains such as intervals over the real numbers, an indication of whether the underlying differential utility function is constant, increasing, or decreasing. In all issues, parameter weights must be nonnegative and sum to 1. The default value for the weights is $1/n$ where n is the number of parameters in the issue.

In the case of parameters taking values from a domain of discrete values, the issue requires an associated specification of a partial ordering on this domain. To maximize (minimize) such a parameter means to prefer dominating (dominated) values from its domain. Suppose, for example, we have a parameter which takes its value from the set

$$S = \{ 6001, 7001, 7002, 8581, 8582, 9520 \}.$$

A possible preference ordering is shown in Figure 1. 9520 is the preferred value when maximizing; 6001 when minimizing.

An example issue would be to minimize parameter `prod_eqpt_cost` with relative weight 0.5 using constant differential utility, minimize parameter `prod_time` with weight 0.3 using decreasing utility, and maximize parameter `mem_cache` with domain S according to the ordering in Figure 1 with weight 0.2.

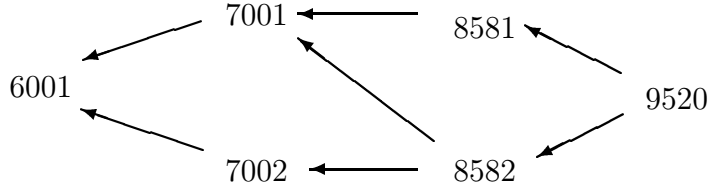


Figure 1: A Partial Ordering on Six Items

3.4 Effects of plans on issues

All client decisions, including the application of a plan, potentially result in changing the value of one or more parameters. A plan P **affects** an issue I if P changes the value assignment of at least one parameter in I . A plan may affect several issues. A plan **negatively affects** an issue through a parameter if the direction of change proposed by the plan for the parameter value is opposite to the desired direction of change defined for the parameter in the issue definition. A plan **positively affects** an issue through a parameter if the direction of change proposed by the plan for the parameter value is the same as the desired direction of change defined for the parameter in the issue definition. For example, suppose a plan P_1 causes the value of the parameter `prod_eqpt_cost` mentioned above to increase. Since the client wishes to minimize this parameter, P_1 is said to negatively affect the issue through parameter `prod_eqpt_cost`. Suppose, on the other hand, a plan P_2 suggests changing the value of `mem_cache` from 7001 to 8582. Since 8582 dominates 7001 in the ordering, P_2 positively affects the issue through `mem_cache`. By the same token, a plan to substitute 8582 for 8581 would not affect the issue through parameter `mem_cache` because those two values are not related in the partial order of Figure 1. The same plan may affect an issue both positively and negatively, through different parameters.

The **partial degree of effect** of a plan on an issue through a parameter is a measure of the impact of the plan on the parameter value compared to the desired direction of change specified in the issue, taking into account the type of differential utility function of the parameter where that function is well-defined. Partial degrees of effect range from -1.0, indicating a maximally negative effect, to +1.0, indicating a maximally positive effect. If an issue mentioning a parameter j is not affected by a plan through j , the partial degree of effect of the plan through j is zero.

The **degree of effect** of a plan on an issue is a measure of the impact of the plan through the set of parameters in the issue. It is a linear combination of the partial degrees of effect through the parameters in the issue, where the weights are the ones given for each parameter in the issue definition. These degrees also occur in the range from -1.0, indicating a very negatively affected issue, to +1.0, indicating a very positively affected issue.

A plan P negatively affects an issue I if the degree of effect of P on I is negative. A plan P positively affects an issue I if the degree of effect of P on I is positive.

A scheme that compares the impact of different plans on a set of issues must not only take into account the weight of each of the parameters in the issue, but also recognize whether the issue is affected by the plan slightly or seriously, positively or negatively. Obviously it is better to choose a plan that slightly negatively affects a given issue rather than a plan that very negatively affects an issue.

The **overall score** of a plan is the computation of its effect on the most negatively affected issue; it is, therefore, a function of the degrees of effect of the issues of all the affected clients. The higher the overall score, the likelier the plan is to be selected. The plan with the highest overall score is selected for use by the clients. Ties are broken by arbitration.

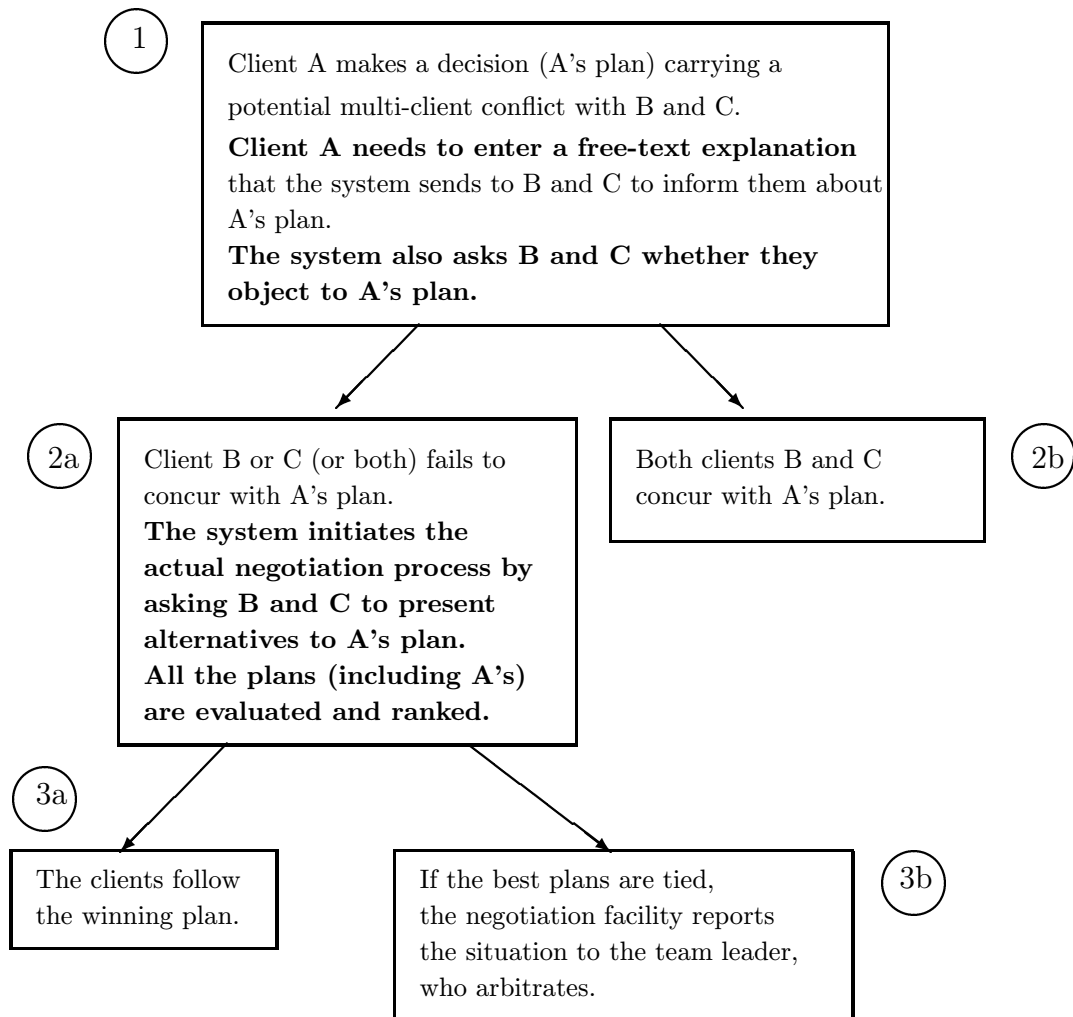


Figure 2: Negotiation Process With Our Protocol

3.5 Evaluation of plans

The **Plan Evaluator** is the evaluation module of the negotiation facility in our protocol. There are two types of situations where the Plan Evaluator could be invoked: a **negotiation** involving multiple clients, which is the main point of this paper; and **prioritization in advice generation**, involving a single client.

The Plan Evaluator computes the effect of a set of alternative competing plans on the issues those plans affect. It requires issues from its clients in order to evaluate the payoff to each client with respect to a plan. Each group of clients with a given perspective (design team, production engineers team, etc.) submits its issue, in the form of a set of weighted preferences involving the parameters that are important from their perspective. These issues are used throughout a design effort for the evaluation of plans in negotiation situations.

The main function of the Plan Evaluator is to support negotiation. The need for negotiation may arise when a client adds or retracts one or more constraints or parameters, when this has the effect of changing the value assignment of at least one parameter in an issue of another client. We call this a **potential multi-client conflict**. Note that this is not the same as a constraint violation. For example, suppose a client A makes a decision (in other words, applies a plan) that causes a potential multi-client conflict between clients B, C, and himself. The system requires a free-text explanation from A about the change he wishes to make. The clients potentially affected by A's plan are asked whether they object to the plan. Without our negotiation protocol, if one of the potentially affected clients (B or C) decided to reject the change or to make a new change, which in turn may affect another client, an endless succession of changes could occur, and the clients might never reach a compromise. Our negotiation protocol, by contrast, is designed to help preclude this type of thrashing. Figure 2 provides an overview of the protocol; significant aspects are shown in boldface.

If at least one client fails to concur with the change, the negotiation process is initiated. The system asks each potentially affected client to present a plan for a compromise within a set period of time. After this time has elapsed, all submitted plans are evaluated by the Plan Evaluator and ranked. The plan which minimizes the dissatisfaction of every client obtains the highest score and is output to the clients.

If two plans get the same score and the Plan Evaluator cannot designate a single best plan, it informs the team leader so that he may arbitrate the situation. Note that, if a client does not respond by the deadline, by default the system assumes that he concurs with the change.

3.5.1 Determination of the affected issues

Whatever the source of suggested changes, client issues may be affected, and determining which issues are affected involves first finding all issues in which one or more affected parameters appear.

For example, consider a plan for changing the tuple of value assignments from

$$(p_1 \leftarrow v_1, p_2 \leftarrow v_2, \dots, p_{10} \leftarrow v_{10})$$

to

$$(p_1 \leftarrow v_1, p_2 \leftarrow v'_2, \dots, p_{10} \leftarrow v'_{10})$$

where the p_i 's are the parameters and the v_i 's are their assigned values. Further, suppose p_2 is one of the members of a frame-like structured domain [Bowen & Bahler 1992] contained hierarchically within another parameter p_{11} . Then the suggested parameter changes in the plan are:

- p_2 would be assigned a new value v'_2 ,

- p_{10} would be assigned a new value v'_{10} .

Therefore, the affected issues are any that contain p_2 or p_{10} , including p_{11} .

3.5.2 Computation of the degrees of effect

Simply knowing which issues are affected is insufficient. The Plan Evaluator also needs to know how much a plan affects each of the affected issues; that is, it needs to compute the degrees of effect on each issue of each plan. The degree of effect of a plan on an issue depends on its effect on the parameters mentioned in the issue, and on the nature of these parameters.

A **desired direction of change** expresses whether a client wishes to minimize or to maximize the value of a parameter in order to improve his own utility. For a parameter taking values from bounded totally-ordered domains, such as numerical intervals, note that identical changes of the same magnitude can constitute either a slight or serious change for a client depending on the domain of the parameter, on where in the domain the old and new values lie, and on whether the client attaches the same importance to a unit of change throughout the domain. We can represent the variable importance associated with the various regions of a parameter's domain if we turn to economics to consider the rate and direction of change of an individual client's utility with different amounts of a good. We deal with three main cases depending on the type of **differential utility function**: constant, increasing, or decreasing. If the differential utility function for a parameter X is constant, a change of X of a given magnitude has the same effect on utility no matter what the value of X . This is the default. If the differential utility function is increasing, the same change of value of a parameter produces a relatively larger change in utility near the maximum value of the domain than near the minimum. A good example may be safety in a nuclear power plant. If the parameter X represents some kind of danger probability, the safety engineer may not be as worried when X goes from 10 to 20% as when it goes from 80 to 90%. Similarly, if the differential utility function is decreasing, the same change of value of a parameter produces a relatively larger change in utility near the minimum value of the domain than near the maximum. Consider the case of buying a computer system, and assume utility is measured solely by price. A change from \$1000 to \$2000 doubles the price whereas a change from \$8000 to \$9000 is just a 12.5% increase.

For parameters on unbounded domains and/or partially-ordered domains, we have not found a principled way to make such a distinction and we differentiate only between desirable and undesirable changes in parameter values.

The goal of the quantitative comparison of plans is to minimize the dissatisfaction of every client rather than maximize the overall satisfaction of the set of clients. The adoption of a plan requires the acceptance of all the members of the team; in other words, enough dissatisfaction on the part of a single client can block the adoption of a plan. Therefore, a plan that slightly lowers the utility of several clients will be preferred to a plan that dramatically lowers the utility of a single client.

We define sgn_{ij} as -1 if the change of value of parameter j in issue i due to the plan is opposite to the desired change, and as $+1$ otherwise. For example, if the difference $\Delta j = j_{new} - j_{old}$ is negative and the issue indicates that the client wants to maximize the parameter, or if the difference Δj is positive and the issue indicates that the client wants to minimize the parameter, then $sgn_{ij} = -1$ and the partial degree of effect is negative; otherwise $sgn_{ij} = +1$ and the partial degree of effect is positive.

Bounded Totally-Ordered Domains. The first step in determining the effect of a plan is the computation of partial degrees of effect for all the parameters whose values are changed by the plan. Since a parameter may appear in several issues, several partial degrees of effect may need to be computed for each parameter. We denote the partial degree of effect of a plan on an issue i

through a parameter j by f_{ij} , and in the case of bounded totally-ordered domains we distinguish three cases depending on the type of differential utility function associated with the parameter.

Constant differential utility function (const_duf): If the differential utility is constant, the partial degree of effect f_{ij} for the j^{th} parameter of issue i is the absolute change in j normalized over its domain $[j_{min}, j_{max}]$. Thus:

$$f_{ij} = sgn_{ij} \times \frac{|\Delta j|}{j_{max} - j_{min}}$$

Increasing differential utility function (incr_duf): For this category, the partial degree of effect is a normalization of the change with respect to the maximum value of the domain:

$$f_{ij} = sgn_{ij} \times \frac{|\Delta j|}{j_{max} - \min(j_{old}, j_{new})}$$

Decreasing differential utility function (decr_duf): For this category, the partial degree of effect is a normalization of the change with respect to the minimum value of the domain:

$$f_{ij} = sgn_{ij} \times \frac{|\Delta j|}{\max(j_{old}, j_{new}) - j_{min}}$$

The normalization of these expression means that $|f_{ij}| \leq 1$ in all cases. From the partial degrees of effect, the Plan Evaluator can compute an overall degree of effect of the plan on each affected issue. The overall degree of effect is simply a linear combination of the partial degrees of effect of the parameters of an issue.

Unbounded Domains. For plans calling for changes to parameters on unbounded domains, we cannot use the formulae of partial degrees of effect given above. In these cases we evaluate these plans by using the weights α_{ij} and the values sgn_{ij} of all the affected parameters. This method applies also to plans containing inequality constraints, or nonspecific plans for increasing or decreasing the value of a numeric parameter without specifying to which new value.

Partially Ordered Domains. In case of parameters over partial orders, sgn_{ij} is computed with respect to the underlying ordering, and plans are evaluated using the weights α_{ij} and the values sgn_{ij} of all the affected parameters.

3.5.3 Overall score computation

Finally, overall scores can be determined for each plan. The **overall score** of a plan P is the lowest degree of effect among the issues affected by P :

$$\min_i \sum_{j \in J_i} (\alpha_{ij} * ((1 - K_j) * f_{ij} + K_j * sgn_{ij})) \quad (1)$$

where i ranges over each affected issue; J_i is the set of all parameters of issue i ; j represents each parameter in each issue; α_{ij} is the weight of parameter j in issue i ; f_{ij} is the partial degree of effect of parameter j on issue i ; and $K_j = 0$ if parameter j has a bounded totally-ordered domain and 1 otherwise.

In the case of plans to increase or decrease the value of a parameter without specifying to which new value, or suggestions affecting unbounded or non-numeric parameters, we evaluate these plans by using the weights α_{ij} and the value of sgn_{ij} for all the parameters of an affected issue. It is worth noting, however, that evaluation may not be fair in this case, because the chosen plan may not have the best overall score for *every possible* change in every parameter.

$\sum_{j \in J_i} (\alpha_{ij} * ((1 - K_j) * f_{ij} + K_j * sgn_{ij}))$ is the plan's degree of effect on issue i . The overall score represents the differential utility due to the plan for the most negatively affected issue (or least positively affected issue, if all effects are positive). Once the overall score of each plan has been computed, the system compares them and recommends for adoption the plan with the highest score.

3.6 Plans and issues in advice generation

Suppose a client alters a parameter lying in her perspective and suppose further that no other perspectives are involved. In this case, if a contradiction occurs it will not require negotiation (at least not immediately) because it involves a single client. This is a violation of a constraint which references parameters in only one perspective. If this happens the system generates a set of suggestions of what the client can do. In a client-assisted system it is highly desirable to provide these system-generated suggestions in the order of the best suggestion first. To do this, relative priority of the suggestions must be evaluated based on how much they affect the client's issue. The system would proceed internally as shown in Figure 3.

4 Example scenarios

Suppose we have a design team involved in the development of a new cache memory hierarchy, and that three perspectives are represented: configuration, production, and test.

The language and run-time system we have designed in which to build design advisors allows for the declaration of issues. In an `issue` statement, the left hand side contains the keyword `issue` followed by a long synonym and a short name for the issue, followed by the keyword `for` and the name of the field of view with which it is associated. The right hand side lists parameters (or domains of parameters) which are involved in the issue with the desired direction of change (`min` or `max`), their respective individual weights within that issue, and, in cases when the differential utility function `duf` is well-defined, an indication of whether it is constant, increasing, or decreasing.

Now suppose the configuration designers set forth this issue:

```
issue 'functionality'(func) for configuration ::=
{ min(no_modules) * 0.5 using decr_duf,
  min(pwr_consump) * 0.3 using incr_duf,
  max(cache) * 0.2 }.
```

and have defined the partial order on parameters of domain `cache` that corresponds to Figure 1:

```
domain(cache) ::= { 9520, 8582, 8581, 7002, 7001, 6001 }.
```

```
prefer(cache) ::= { 9520 > (8581,8582), 8582 > (7001,7002),
  8581 > 7001, 7002 > 6001, 7001 > 6001 }.
```

Suppose the production engineers have expressed this issue:

```
issue 'manufacturability'(manuf) for production ::=
{ min(no_chip_areas) * 0.4 using const_duf,
```

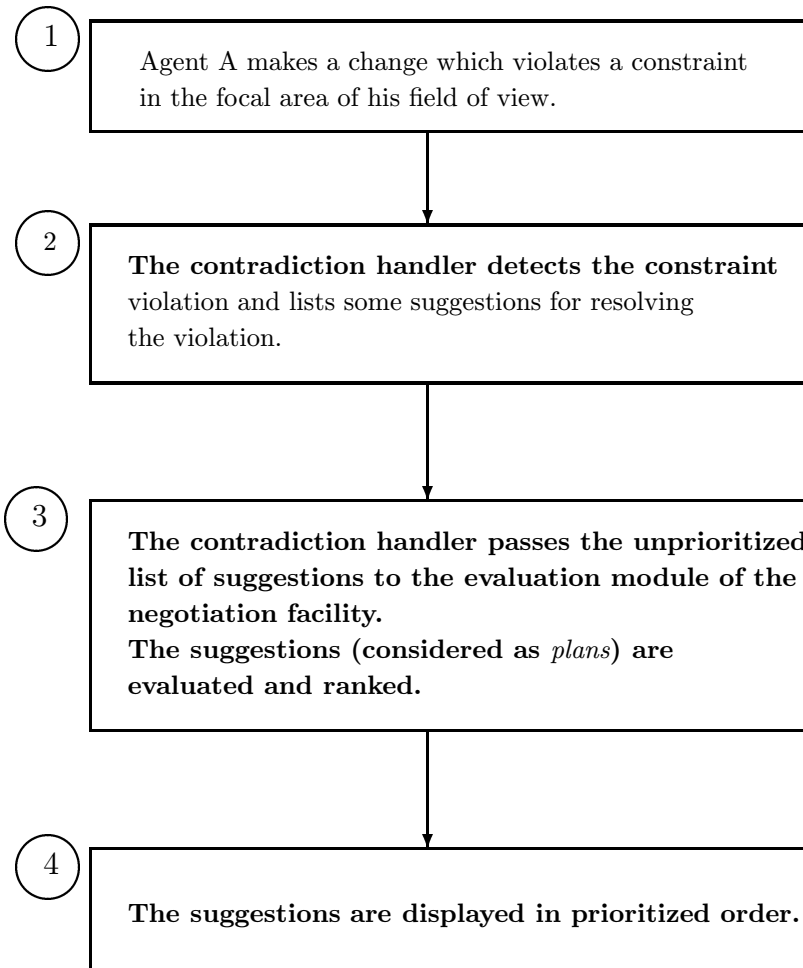


Figure 3: Prioritization of System Suggestions Using Our Protocol

```

max(eqpt_utilization) * 0.2 using decr_duf,
min(prod_eqpt_cost) * 0.4 using decr_duf }.

```

Finally, suppose the test engineers have this issue:

```

issue 'testability'(test) for test ::=
{ min(test_eqpt_cost) * 0.1 using decr_duf,
  min(no_modules) * 0.2 using const_duf,
  min(changeovers) * 0.7 }.

```

4.1 Scenario 1: Potential multi-client conflict

In this scenario, the configuration designers will make a decision causing a potential multi-client conflict with the production engineers, so that negotiation is needed. The suggested compromise from the production engineers will provoke a potential conflict with the test engineers, who in turn will suggest a compromise impacting the configuration designers.

Suppose the designers want to modify the set of modules in a cache memory hierarchy from 7001 and 7002 to the single module 8582 in order to reduce the number of modules (minimize the parameter `no_modules`) and to move upward in their preference ordering for parameters of domain `cache`. Suppose the modules 7001 and 7002 use the same chip areas, whereas the module 8582 requires a different chip area.

Therefore, the number of different chip sizes required would increase with the designers' decision, say from 1 to 2, and this change would negatively affect the parameter `no_chip_areas` within the production engineers' issue `manufacturability`. Note that this effect does not require introducing a constraint violation.

The system requires an explanation from the designers to justify their decision. The designers explain that they want to minimize the number of modules by replacing the modules 7001 and 7002 by the module 8582. The system informs the production engineers about the designers' intention of doing this, along with the explanation. The system also asks the production engineers whether they concur with this decision, considering that the decision would increase the number of required chip areas from 1 to 2.

Suppose the production engineers do not concur with the decision and are therefore asked to submit a plan for a compromise with the designers. The production engineers suggest replacing the modules 7001 and 7002 by the module 8581 rather than by the module 8582, because let us say the module 8581 uses the same chip area as the module 9520 which is already in the design. Suppose module 8581, however, would increase power consumption from 100 to 200 units. More importantly, this plan impacts the test perspective by increasing number of changeovers necessary in the test equipment (`changeovers`).

Because of this impact, the test engineers are now invited to offer a plan. Suppose they suggest substituting module 6001 for 8581, which does not require increased changeovers but which is dominated in the configuration designers' ordering by all other module choices.

The configuration designers' decision constitutes Plan 1, the production engineers' counter-suggestion constitutes Plan 2, and the test engineers' rejoinder is Plan 3. Figure 4 summarizes the computation of the overall scores of the three plans.

Plan 1 affects two issues: `functionality` and `manufacturability`. Within the issue `functionality` only one parameter is affected: `no_modules`, so the degree of effect of the issue `functionality` simply equals the product of the weight of parameter `func.no_modules` by its partial degree of effect.

$$f_{ij} = sgn_{ij} * \frac{|\Delta j|}{\max(j_{old}, j_{new}) - j_{min}}$$

which here is equivalent to:

$$f_{ij} = (+1) * \frac{|-1|}{\max(3, 2) - 0} = +0.33$$

. Thus, since $\alpha_{ij} = 0.5$, the degree of effect of Plan 1 on the issue `functionality` rounds to +0.165.

	Affected Parameters	Δj	Domain	Desired dir. of change	sgn_{ij}	Type of d.u.f.	f_{ij}	α_{ij}	$\alpha_{ij} * f_{ij}$
Plan 1	func.no_modules	-1	[0..100]	min	+1	decr_duf	+0.33	0.5	+0.165
	manuf.no_chip_areas	+1	[0..20]	min	-1	const_duf	-0.05	0.4	-0.02
Overall score for Plan 1	$\min(+0.165, -0.02) = -0.02$								
Plan 2	func.no_modules	-1	[0..100]	min	+1	decr_duf	+0.33	0.5	+0.5
	func.pwr_consump	+100	[0..500]	min	-1	incr_duf	-0.25	0.3	-0.3
	test.changeovers	+100		min	-1			0.7	-0.7
Overall score for Plan 2	$\min((0.5 + (-0.3)), -0.7) = -0.7$								
Plan 3	func.no_modules	-1	[0..100]	min	+1	decr_duf	+0.33	0.5	+0.5
	func.cache			max	-1			0.2	-0.2
Overall score for Plan 3	$0.5 + (-0.2) = 0.3$								
Output	Plan 3 selected								

Figure 4: Scenario 1 – Evaluation of Plans From Multiple Clients

Similarly, we compute the degree of effect of Plan 1 on the issue manufacturability, in which only the parameter `manuf.no_chip_areas` is affected.

$$f_{ij} = sgn_{ij} * \frac{|\Delta j|}{j_{max} - j_{min}}$$

which here is equivalent to:

$$f_{ij} = (-1) * \frac{|+1|}{20 - 0} = -0.05$$

. Thus, since $\alpha_{ij} = 0.4$, the degree of effect of the issue `manufacturability` rounds to -0.02 , and the overall score for Plan 1 is $\min(+0.165, -0.02) = -0.02$.

Plan 2 affects two issues, `functionality` and `testability`, and within issue `functionality` two parameters are affected: `no_modules` and `pwr_consump`. With issue `testability`, parameter `changeovers` is affected. Thus the score for Plan 2 is the minimum of its effect on `functionality` and its effect on `testability`, where its effect on `functionality` is the sum of two terms: (i) the degree of effect through `func.no_modules` (+0.5), which is the product of the weight of `func.no_modules` and its partial degree of effect due to Plan 2, and (ii) the degree of effect through `func.pwr_consump` (-0.3), which is the product of the weight of `func.pwr_consump` and its partial degree of effect due to this plan. Plan 2's effect on `testability` is -0.7 through `test.changeovers`. The score for Plan 2 rounds to -0.7 .

Plan 3 affects only one issue, `functionality`. Its overall effect is the sum of its effect on `func.no_modules` (0.5) and its effect on `func.cache` (-0.2), or 0.3.

Plan 3 has the highest score. Therefore it is selected by the Plan Evaluator as the best plan, and the designers are asked by the system to replace the modules 7001 and 7002 by the module 6001.

4.2 Scenario 2: Evaluation of nonspecific plans

The method of evaluating plans can also be used in the case of suggestions for increasing or decreasing the value of a parameter without specifying to which new value. Here again we evaluate these nonspecific plans by using the weights α_{ij} and the values sgn_{ij} of all the affected parameters. Suppose we have the same issues and initial situation as in Scenario 1, and that, at some point, a client faces a conflict and two suggestions have been offered to help remedy it. Suppose that this time only the proposed direction of change for one affected parameter is known. Here are the assumed effects of both suggestions:

	Affected Parameters	Δj	Domain	Desired dir. of change	sgn_{ij}	Type of d.u.f.	f_{ij}	α_{ij}	$\alpha_{ij} * sgn_{ij}$
Plan 1	func.no_modules	+3	[0,100]	min	-1	decr_duf		0.6	-0.6
	manuf.no_chip_areas	+1	[0,20]	min	-1	const_duf		0.2	-0.2
	manuf.eqpt_utilization	increase	[0,100]	min	-1	const_duf	n/a	0.4	-0.4
Overall score for Plan 1	$\min(-0.6, ((-0.2) + (-0.4))) = -0.6$								
Plan 2	func.no_modules	-1	[0,100]	min	+1	decr_duf		0.6	+0.6
	manuf.no_chip_areas	+2	[0,20]	min	-1	const_duf		0.2	-0.2
	manuf.prod_eqpt_cost	+10	[0,100]	max	+1	decr_duf		0.2	+0.2
Overall score for Plan 2	$\min(+0.6, ((-0.2) + (+0.2))) = 0$								
Output	Plan 2 selected								

Figure 5: Scenario 2 – Evaluation of Nonspecific Plans

- Plan 1 would affect three parameters:
 - no_modules, whose value would increase by 3,
 - no_chip_areas, whose value would increase by 1, and
 - eqpt_utilization, whose increase is not specified.
- Plan 2 would affect an overlapping set of three parameters:
 - no_modules, whose value would decrease by 1,
 - no_chip_areas, whose value would increase by 2, and
 - prod_eqpt_cost, whose value would increase by 10.

Plan 2 is a specific plan because all its suggested changes are fully specified. But, since Plan 1 does not specify how much the client should increase parameter `eqpt_utilization`, Plan 1 is nonspecific, and the degree of effect for each issue is computed by:

$$\sum_{j \in J_i} (\alpha_{ij} * sgn_{ij}).$$

As shown in Figure 5, Plan 2 wins, since its score is 0 versus -0.6 for Plan 1.

4.3 Scenario 3: Prioritization in single-client advice generation

The third scenario illustrates another type of situation in which the Plan Evaluator could be invoked: the prioritization of suggestions generated by the system’s contradiction handler before the display of suggestions for a constraint violation resolution. This situation involves a single perspective, that of production engineers.

Suppose the team of production engineers encounters a contradiction involving constraints C_a and C_b within the focal area of its perspective, and no other perspectives are involved. The system detects the contradiction and offers a choice of two suggestions: either retract constraint C_a and thereby change the value of parameter `no_chip_areas` from 2 to 7 (call this Plan 1), or retract constraint C_b and thereby change the value of parameter `eqpt_utilization` from 80 to 70 (call this Plan 2). The plans need to be compared in order to provide prioritized suggestions to the production engineers and help them select the plan that gives them the greater utility. Since each plan affects only a single issue, their scores are easy to compute; the computation is shown in Figure 6. Since Plan 2 (+0.04) has greater utility than Plan 1 (-0.01), the system’s suggestion generator would display this list of prioritized suggestions:

- (1) retract constraint C_b
- (2) retract constraint C_a

	Affected Parameters	Δ_j	Domain	Desired dir. of change	sgn_{ij}	Type of d.u.f.	f_{ij}	α_{ij}	$\alpha_{ij} * f_{ij}$
Plan 1	manuf.no_chip_areas	+5	[0..100]	min	-1	const_duf	-0.05	0.5	-0.025
Overall score for Plan 1	-0.025								
Plan 2	manuf.eqpt_utilization	-10	[0..100]	min	+1	const_duf	+0.1	0.2	+0.02
Overall score for Plan 2	+0.02								
Prioritized suggestions	1. Select Plan 2 2. Select Plan 1								

Figure 6: Scenario 3 – Prioritization in Advice Generation

5 Comparison with other work

Wellman [Wellman 1994] has presented a market-based model for distributed configuration design problems, which employs a computational economy of producers and consumers to allocate resources to design participants. Although these design economies are not guaranteed to find optimal designs, market exchange of resources is an interesting paradigm for the process of resource trading that takes place in many design efforts. However, the market model lacks the expressiveness to represent realistic design knowledge and constraints in some cases, and so would need to be supplemented with a companion constraint-directed reasoning system or hybridized with some other approach.

The Designer Fabricator Interpreter (DFI) [Werkman 1991, Werkman 1992] is another negotiation model for concurrent engineering based on multiple perspectives. Agents combine their issues in sharable perspectives. The DFI system allows a user to present a proposal, along with a key issue, to the entire group for evaluation. The key issue represents the interests of an agent and must be respected by all the other agents throughout the negotiation dialogue. The proposal is reviewed and refined into counter-proposals. Once all generated proposals have been evaluated, the best one is selected. This mechanism facilitates the negotiation process because it privileges the agent with the key issue with respect to the others, but it may not be considered fair.

Case-based negotiation systems use an organized memory of previous cases of conflicts to automatically generate a solution for the current conflict. The system analyzes the differences between the past case, selected from the memory, and the current case in order to modify the past solution appropriately. PERSUADER [Sycara 1989] is a system that models human behavior during adversarial labor-management negotiation. PERSUADER uses case-based reasoning to resolve multi-agent conflicts, and uses preference analysis to select a compromise solution when case-based reasoning cannot be applied. In preference analysis, the system approximates the shape of the utility curves for each agent. The utility curve shapes are common information among agents, which helps each agent predict whether a solution is going to satisfy the other agents by giving them adequate payoff.

Cosmos is a tool which supports negotiation among designers by determining the consequences of a change suggested by a designer and providing that information to the whole group of designers. The system thus supplies a common context which the team can use as a basis for discussion. Cosmos is intended as a tool in the large Palo Alto Collaborative Testbed (PACT) [Cutkosky *et al.* 1993]. Another aspect of this project is SHADE [Gruber *et al.* 1992, Tenenbaum *et al.* 1992], a representational framework for sharing design knowledge and coordinating the communication of design changes.

Other systems support collaborative design but are not oriented toward mediating negotiation explicitly. REDUX' [Petrie 1992] is a decision maintenance server for distributed design tasks that maintains dependencies among objects, where each object is of a type defined in a shared ontology of decision components. Clients send to REDUX' messages about decisions and related information. The output is messages representing the propagation of user design changes. Although REDUX' represents knowledge in constraint form, it performs coordination services only; the users are responsible for detecting constraint violations and determining which decisions need to be altered to resolve the conflict. REDUX' then uses truth maintenance techniques to propagate the consequences consistently. The ADD system [Garcia 1992] supports parametric design by

using representation of both constraint information and evaluation criteria representing designer preferences. The system allows the user to modify or delete criteria, change the alternative evaluation of a criteria, or change the overall importance played by a criterion in a design. The system itself, however, does not appear to have a mechanism for manipulating these criteria, or comparing conflicting criteria among users.

6 Conclusions

Our negotiation protocol provides a well-founded, expressive, and useful support for negotiation between the various users of client-assisted inference systems for concurrent engineering.

The protocol not only detects when a client makes a decision potentially affecting other clients, it also offers a procedure for evaluating the various solution plans that clients suggest. This evaluation is based on, first, the issues entered by the different members of the team before or during the design process, and second, the different plans suggested by the affected clients for finding a compromise. The protocol works even for non-numeric parameters: knowing only the direction of change of the affected parameters is sufficient to determine the degree of effect of the plan for the issues containing the affected parameters.

We recognize that generation of viable alternatives which can then be subjected to our form of arbitration is itself a major challenge in the support of collaborative design. For examples of how our system addresses these issues, see [Bahler & Bowen 1995, Bowen & Bahler 1991a].

The protocol presented here uses a fair evaluation scheme, in the sense defined above. First, it treats all the clients equally. Second, it avoids any situation where a client could be greatly dissatisfied because several other clients see their position slightly improved by the acceptance of a given plan. Finally, no issue has priority over any other issue.

Future plans include completing the implementation of the methods and evaluating them experimentally.

7 Acknowledgments

This work was partially supported by the National Science Foundation under Grant No. DDM-9215755 and by IBM Corp. We also wish to thank the anonymous reviewers for comments which helped improve the presentation of this material.

References

- [Bahler & Bowen 1995] Bahler, D. and Bowen, J. 1995. Constraint Logic and Its Application in Production: An Implementation Using the Galileo4 Language and System. In A. Artiba and S.E. Elmaghraby (eds.), *Production and Scheduling of Manufacturing Systems*, London: Chapman and Hall.
- [Bahler *et al.* 1994b] Bahler, D., Dupont, C., and Bowen, J. 1994. Mediating Conflict in Concurrent Engineering with a Protocol Based on Utility. *Concurrent Engineering: Research and Applications*, Special Issue on Conflict Management in Concurrent Engineering.
- [Bahler *et al.* 1994a] Bahler, D., Dupont, C., and Bowen, J. 1994. An Axiomatic Approach That Supports Negotiated Resolution of Design Conflicts in Concurrent Engineering. In J.S. Gero (ed.), *Artificial Intelligence in Design*, Dordrecht, The Netherlands: Kluwer Academic Publishers.
- [Bahler *et al.* 1993] Bahler, D., Dupont, C., and Bowen, J. 1993. Negotiation in client-assisted inference systems for Concurrent Engineering. *Workshop on Computational Models of Conflict Management in Cooperative Problem Solving*, Chambery, France, 23-40.
- [Bowen & Bahler 1991a] Bowen, J. and Bahler, D. 1991. Conditional Existence of Variables in Generalized Constraint Networks. *Proc. of 9th Nat. Conf. on AI (AAAI-91)*, Anaheim CA, 215-220.

- [Bowen & Bahler 1991b] Bowen, J. and Bahler, D. 1991. Supporting Cooperation between Multiple Perspectives in a Constraint-based approach to Concurrent Engineering. *Journal of Design & Manufacturing* **1**, 89-105.
- [Bowen & Bahler 1992] Bowen, J. and Bahler, D. 1992. Frames, quantification, perspectives and negotiation in constraint networks for life-cycle engineering. *Int. Journal of AI in Engineering* **7**, 199-226.
- [Bowen & Bahler 1993] Bowen, J. and Bahler, D. 1993. Constraint-Based Software for Concurrent Engineering. *IEEE Computer* **26**, Special Issue on Computer Support for Concurrent Engineering.
- [Cutkosky *et al.* 1993] Cutkosky M., Engelmores R., Fikes R., Gruber T., Genesereth M., Mark W., Tenenbaum J., and Weber J. 1993. PACT: An Experiment in Integrating Concurrent Engineering Systems. *IEEE Computer* **26**, Special Issue on Computer Support for Concurrent Engineering.
- [Daboni *et al.* 1986] Daboni, L., Montesano, A., and Lines, M. (Eds.) 1986. *Recent Developments in the Foundations of Utility and Risk Theory*, Dordrecht, The Netherlands: D. Reidel.
- [Erman *et al.* 1988] Erman, L.D., Lark, J.S. and Hayes-Roth, F. 1988. ABE: An Environment for Engineering Intelligent Systems. *IEEE Transactions on Software Engineering* **SE-14**, 1758-1769.
- [Fishburn 1988] Fishburn, P. C. 1988. *Nonlinear Preference and Utility Theory*, Baltimore: Johns Hopkins University Press.
- [Garcia 1992] Garcia, A.C.B. and H.C. Howard 1992. ADD's Model for Acquiring Design Rationale. *Proc. AAAI '92 Workshop on Design Rationale Capture and Use*, San Jose CA, 91-98.
- [Gasser *et al.* 1987] Gasser, L., Braganza, C. and Herman, N. 1987. MACE: A Flexible Testbed for Distributed AI Research. In M.N. Huhns (ed.), *Distributed Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann, 119-149.
- [Gruber *et al.* 1992] Gruber, T.R., Tenenbaum, J.M., and Weber, J.C. 1992. Toward a Knowledge Medium for Collaborative Product Development. In J.S. Gero (ed.), *Artificial Intelligence in Design '92*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 413-432.
- [Hillier & Lieberman 1990] Hillier, F.S. and Lieberman, G. J. 1990. *Introduction to Operations Research*, New York: McGraw Hill.
- [Keeney & Raiffa 1976] Keeney, R.L. and Raiffa, H. 1976. *Decisions with Multiple Objectives*, New York: Wiley.
- [Petrie 1992] Petrie, C. 1993. The REDUX' Server. *Proc. Intl. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam.
- [Rosenschein & Genesereth 1985] Rosenschein, J.S. and Genesereth, M.R. 1985. Deals among Rational Agents. *Proc. 9th Int. Joint Conf. on AI (IJCAI-85)*, Los Angeles, 91-99.
- [Samuelson & Nordhaus 1992] Samuelson, P.A. and Nordhaus, W.D. 1992. *Economics*, 14th ed. New York: McGraw Hill.
- [Sycara 1989] Sycara, K.P. 1989. Multiagent compromise via negotiation. In L. Gasser and M. N. Huhns, eds., *Distributed Artificial Intelligence* **2**. London: Pitman/Morgan Kaufmann, 119-138.
- [Tenenbaum *et al.* 1992] Tenenbaum, J., Weber, J., and Gruber, T. 1992. Enterprise Integration: Lessons from SHADE and PACT. In C. Petrie (ed.), *Enterprise Integration Modeling*. Cambridge, MA: MIT Press.
- [Tversky & Kahneman 1981] Tversky, A. and Kahneman, D. 1981. "The Framing of Decisions and the Psychology of Choice," *Science* **211**, 453-58.
- [Werkman 1991] Werkman, K.J. 1991. Using negotiation as a means of coordinating distributed problem solving. *Proc. World Congress on Expert Systems*, Orlando, FL.

[Werkman 1992] Werkman, K.J. 1992. Cooperative design evaluation between multiple agents using negotiation with shareable perspectives. IBM Corporation Monograph.

[Wellman 1994] Wellman, M., "A Computational Market Model for Distributed Configurational Design," *Proc. 12th Nat. Conf. on Artificial Intelligence (AAAI '94)*, 401-407.