

# Mediating Conflict in Concurrent Engineering with a Protocol Based on Utility<sup>1</sup>

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  
University College  
Cork, Ireland

---

<sup>1</sup>This work was supported in part by NSF under Grant No. DDM-9215755 and by IBM Corp.

## Abstract

Concurrent engineering is an approach to product, process, and organizational design in which experts from many disciplines contribute at many stages of the product life cycle. Even though the design process is mostly cooperative, conflicts among sharply diverging viewpoints may occur, which require negotiated compromise solutions. Design advice tools can assist in this process of negotiation by making their critiques and suggestions conveniently available to all members of the product development team. Constraint-based languages of sufficient expressive power provide a convenient representation of real-world problems like those encountered in concurrent engineering, and we believe they are a good basis for building such design advice tools. In this paper we describe a protocol, based on the use of economic utility, by which constraint-based design advice systems can recognize conflict, and support and mediate negotiation fairly.

Keywords: Teaming and Sharing, Reasoning and Negotiation, Collaborative Decision Making

## 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 [3]. 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.

These design advice tools are intended for use by and for multiple clients, including both computer systems and human users. 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 employ concepts from the economic theory of utility to construct a negotiation protocol for such design advice tools. For more detail see [7].

In our experience [3, 4, 5] constraint-based systems constitute a good basis for such tools; much of our earlier effort has emphasized expressive competence in constraint-based languages in order to provide a convenient and sufficient representation of the kind of real-world problems encountered in concurrent engineering. For this reason we illustrate most of our work with constraint-based systems, although we believe our negotiation protocol has wide applicability to any client-assisted inference system.

## 2 The cooperative/competitive duality of negotiation

The members of a concurrent engineering design team act both in a cooperative and competitive fashion, usually exhibiting what has been called the “benevolent agent assumption” [13]. On the one hand, in the end all agents possess a common high-level goal: to design a product with the best possible performance/cost ratio in order to compete successfully in the market. On the other hand, 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 critiques about the proposed design aimed at improving their own interests in the design process. It is a wise assumption that each agent’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 team members, so each agent has to defend his interests and consider the others as adversaries. The potential for conflict and the need for negotiation in such a context are obvious.

Our ambitions in support of negotiation are somewhat different from those of many other investigators. Fully automated negotiation involves computer-based agents and a system able to provide its own compromise proposals, which constitute alternative ways to resolve conflicts. Our approach, by contrast, automates negotiation only partially. In partially automated negotiation, the compromise proposals are supplied by **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 compromise proposals. This might be called **client-assisted negotiation**, by analogy with the more general notion of client-assisted inference [1]. While comparatively more researchers have shown interest in fully automated negotiation [8, 10, 13, 15] than in our form of partially automated negotiation [4, 17, 18], it is our view that partially automated negotiation, and more generally partially automated design, is a more appropriate paradigm 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. Because of this belief, we make no attempt to generate client preferences or compromise proposals automatically, or to try to match each case with a previous experience. On the other hand, the relatively sparse discussions of partially automated negotiation present mostly general guidelines and ad-hoc methods rather than detailed specifications or discussion of actual implementations. This leaves a gap that this work attempts to fill.

### 3 Requirements of a negotiation protocol for concurrent engineering

Assuming a context of client-assisted design-advice systems, it is possible to identify several desirable characteristics of a protocol to support negotiation among concurrent engineering team members. Any negotiation protocol should provide the following support to the team:

1. Collection of client preferences;
2. Detection of situations of potential conflict between design decisions;
3. Collection of suggested compromise solutions;
4. Evaluation based on the client preferences of those solutions; and
5. Output of the best solution to the clients for its execution.

Moreover, a negotiation protocol should 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.

### 4 Galileo4, a client-assisted inference system

Galileo4 [4, 5] is an implemented constraint-based language and run-time system originally developed for building design advisors for concurrent engineering. The main features of Galileo4 are the following:

- Frame-like structured domains can be defined and organized in (multiple) inheritance hierarchies.
- Constraints may be universally and/or existentially quantified, quantifiers being nested in arbitrarily complex ways.
- The existence of certain parameters may be specified as contingent on the truth or falsity of arbitrarily complex conditions. This use of so-called free logic constraints [2] enables Galileo4 to support architectural, as well as parametric, design. (In parametric design, values are assigned to an existing set of parameters; in architectural design, the question of what parameters should belong to that set is itself determined.)
- A form of modal logic is supported, which enables preference orderings and relative priorities to be established over sets of disjunctive constraints.
- Daemons, or forward-chaining imperative productions, can be used to program non-standard treatment of constraint violations or to specify behavior that must be exhibited by the design advisor when the truth of salient constraints is definitely established; the full expressive power of the constraint specification language is available when specifying the condition parts of these daemons while the action parts can invoke procedures written in an imperative sub-language or can execute external programs by making calls to the operating system.
- Explanations are provided to the client based on dependency records that associate each parameter's existence and value assignment with the constraints and parameters on which it depends. Galileo4 uses its dependency records to recover from inconsistent states, to generate advice to its clients, and to provide explanations justifying the existence and value of parameters. These techniques support negotiation by helping to detect inconsistencies that need to be resolved by the negotiation facility.
- Natural language synonyms can be provided for all concepts coded in an application, thereby enabling programs to explain themselves to their clients by providing natural language phrases.
- At run-time, a Galileo4 program provides constraint-processing services for its clients, which may be human interactive users or, in the most recent prototype versions, other systems. Client decisions and system inference may be freely interleaved. All interaction with Galileo4 consists of adding and retracting constraints, including simple constraints such as the equalities whose function is to assign values to design parameters. Galileo4's expressive competence means that its inference is sound and terminating but not necessarily complete, in the sense that the set of solutions for a design problem may be a proper subset of the actual set of solutions computed by Galileo4, but no solutions will lie outside the computed set.

#### 4.1 Multiple perspectives

Galileo4 already includes some syntactic bases for negotiation. To our knowledge Galileo4 and its immediate predecessors are the first constraint languages to allow the definition of **fields of view** (also called perspectives), which are subsets of the design parameters together with their associated constraints.<sup>2</sup> A field of view is defined in a Galileo4 program by specifying all its parameters and

---

<sup>2</sup>A similar notion was subsequently incorporated into the Constraint Management System developed within the DARPA Initiative in Concurrent Engineering.

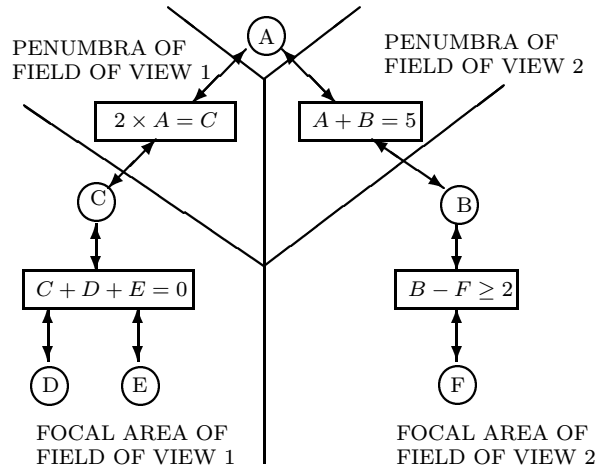


Figure 1: Focal areas and penumbras of two fields of view

the name of the class of clients which can access it, so that for example the perspective of circuit designers might be defined as all parameters of domain (i.e., type) **component**.

Fields of view allow constraint networks to be partitioned into regions reflecting the perspectives of the engineering disciplines involved in the design team, whether it be functionality, manufacturability, testability, marketing, or any other. Fields of view may be organized hierarchically, allowing a client to access one field of view from another.

A field of view in Galileo4 contains both a set of network parameters and all constraints which reference those parameters. We distinguish between the constraints which reference only parameters inside the field of view, which form the so-called **focal area** of the field of view, and the constraints which reference some parameters inside and some outside the field of view, which form the so-called **penumbra** of the field of view. An example is shown in Fig. 1. Field of view 1 contains the parameters  $C$ ,  $D$ , and  $E$ . Its focal area is composed of the parameters  $C$ ,  $D$ , and  $E$ , together with the constraint  $C + D + E = 0$ . The constraint  $2 \times A = C$  is in the penumbra of field of view 1, since  $C$  belongs to field of view 1, but  $A$  does not. Field of view 2 is organized analogously.

Until now, the Galileo languages contained no detailed protocol for using fields of view in a negotiation process. The most common potential need for negotiation arises when a client makes a decision that has the direct or indirect effect of altering a parameter that lies in the field of view of another client. Any other client sharing the parameter is told that a parameter lying in his field of view has been changed, and he may resolve the conflict by altering a design parameter under his control. Since sometimes no such compromise is possible, we set out to extend this framework for negotiation support into a full-fledged negotiation protocol.

## 5 The economic theory of demand and utility

In the event of a conflict, the protocol we describe calls for all involved parties to present compromise solutions; the negotiation facility is then responsible for comparing and ranking the solutions and suggesting the most suitable compromise. The goal of the negotiation facility module responsible for comparing the collected compromise solutions is to select the proposal that can best satisfy all the clients.

The degree of satisfaction provided by a proposal to a client may be captured by the notion of economic **utility** [14]. In economics, utility is the total satisfaction deriving from the consumption

of goods and services. Total utility is a function of the mix of goods and services and also of the importance attached by the consumer to them. Modern treatments of utility define an **indifference curve** (or, more generally, surface) as the set of all combinations of goods that a consumer finds equally desirable. For example suppose a consumer is indifferent to the choice among combinations A, B, C, and D; in this case A, B, C, and D would lie on the same indifference curve. An indifference map is a set of indifference curves (or surfaces) each corresponding to some utility. If batch D would give the consumer more satisfaction than batch E, its curve would lie everywhere further from the utility-space origin than the curve containing E. The typical hyperbolic shape of these curves can be explained by the substitution law: “the scarcer a good, the greater its relative substitution value; its marginal utility rises relative to the marginal utility of the good that has become plentiful.” [14]

The degree of satisfaction of  $n$  consumers (or groups) can be represented by a **utility-possibility frontier** in  $n$  dimensions. In the context that interests us, namely the selection among possible ways to solve a design conflict among multiple clients with different perspectives, the satisfaction of each client (or group of clients) can be represented as an  $n$ -dimensional utility-possibility frontier. The idea is to accept a proposal only if *every* client’s position stays as far from the origin as possible. In the best case, the proposal should allow every client to stay on the same utility-possibility frontier or to move to another frontier further from the origin.

## 6 Our negotiation protocol

This section describes a negotiation protocol offering the following capabilities:

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

### 6.1 Terminology

The vocabulary and the syntax we have selected for describing our protocol is inspired by the formalism underlying Galileo4. Nevertheless, the protocol is applicable to any client-assisted inference system.

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 value assignments for some subset of parameters of  $X$ , where  $\langle U, X, C \rangle$  is a constraint network. Operationally, a plan consists of any action which has the effect of either assigning a new value to a previously unassigned parameter or replacing an existing value with a new one. Such actions may be any of the following:

- *Adding a constraint.* Currently the only constraints permitted in plans are 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 zero, in which case the parameter  $X$  will be assigned the value 0.25.
- *Retracting a constraint.* Assume that we have two constraints referring to a parameter  $X$ :  $X < 1$  and  $X + Y = 2$ , and that the current value for  $Y$  is zero. There is a constraint violation since  $X$  cannot simultaneously equal 2 and be less than 1. If a client retracts the constraint  $X < 1$ ,  $X$  will be assigned the value 2.

A parameter is affected by a plan if its value assignment is modified by the application of the plan. This definition of a plan allows only some of the syntactically legal Galileo4 constraints and relaxing this restriction is a topic of current research (see Sec. 9). Of course, it is always possible that a tuple of values may violate a constraint, either within the original perspective or some other. In a constraint-based system, such contingencies are handled by the regular constraint-violation mechanism, before the negotiation facility can take effect.

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.

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. Note that identical changes of the same magnitude for a parameter 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 return 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.

An **issue** is defined as a set of parameters, each of which is qualified by two attributes, a desired direction of change and a differential utility function. The notion of issue is related to that of a Galileo4 perspective (or field of view) discussed above. 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. We currently restrict ourselves to issues which contain only totally ordered parameters defined over bounded domains, but this is a subject for further research.

The EBNF syntax for an issue is:

```
<ISSUE> ::= issue <ISSUE_NAME> for <FIELD> ::= { <ISSUE_LIST> }
<FIELD> ::= <STRING>
```

```

<ISSUE_NAME> ::= '<LONG_SYNONYMN>' ( <SHORT_NAME> )
<ISSUE_LIST> ::= <ISSUE_BODY> | <ISSUE_BODY>, <ISSUE_LIST>
<ISSUE_BODY> ::= <DESIRED_DIR_OF_CHANGE> ( <PARAMETER_NAME> )
                [ * <WEIGHT> ] [ using <DUF_TYPE> ]
<LONG_SYNONYMN> ::= <STRING>
<SHORT_NAME> ::= <STRING>
<DESIRED_DIR_OF_CHANGE> ::= min|max
<PARAMETER_NAME> ::= <STRING>
<WEIGHT> ::= <REAL_NUMBER>
<DUF_TYPE> ::= const_duf|incr_duf|decr_duf

```

An example definition of an issue is:

```

issue 'production cost effectiveness'(prod_cost_eff)
    for production ::=
    { min(prod_eqpt_cost) * 0.4 using const_duf,
      min(prod_time) * 0.6 using decr_duf }.

```

The left hand side of the statement 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 which are involved in the issue with the desired direction of change (**min** or **max**), their respective individual weights within that issue (0.4 and 0.6) and an indication of whether the underlying differential utility function **duf** is constant, increasing, or decreasing. 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.

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  $P$  causes the value of the parameter **prod\_eqpt\_cost** mentioned above to increase. Since the client wishes to minimize this parameter,  $P$  is said to negatively affect the issue **prod\_cost\_eff** through parameter **prod\_eqpt\_cost**.

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. 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.

Our work may be contrasted to those knowledge-based methods which attempt to model human negotiation and to fully automate the negotiation process, either by using case-based reasoning [15] or some type of planning. Rather than following that line, we opt for an axiomatic protocol. Our method automates the negotiation process only partially, by defining the properties that the solution must satisfy, and then deriving procedures that compute agreement points that satisfy

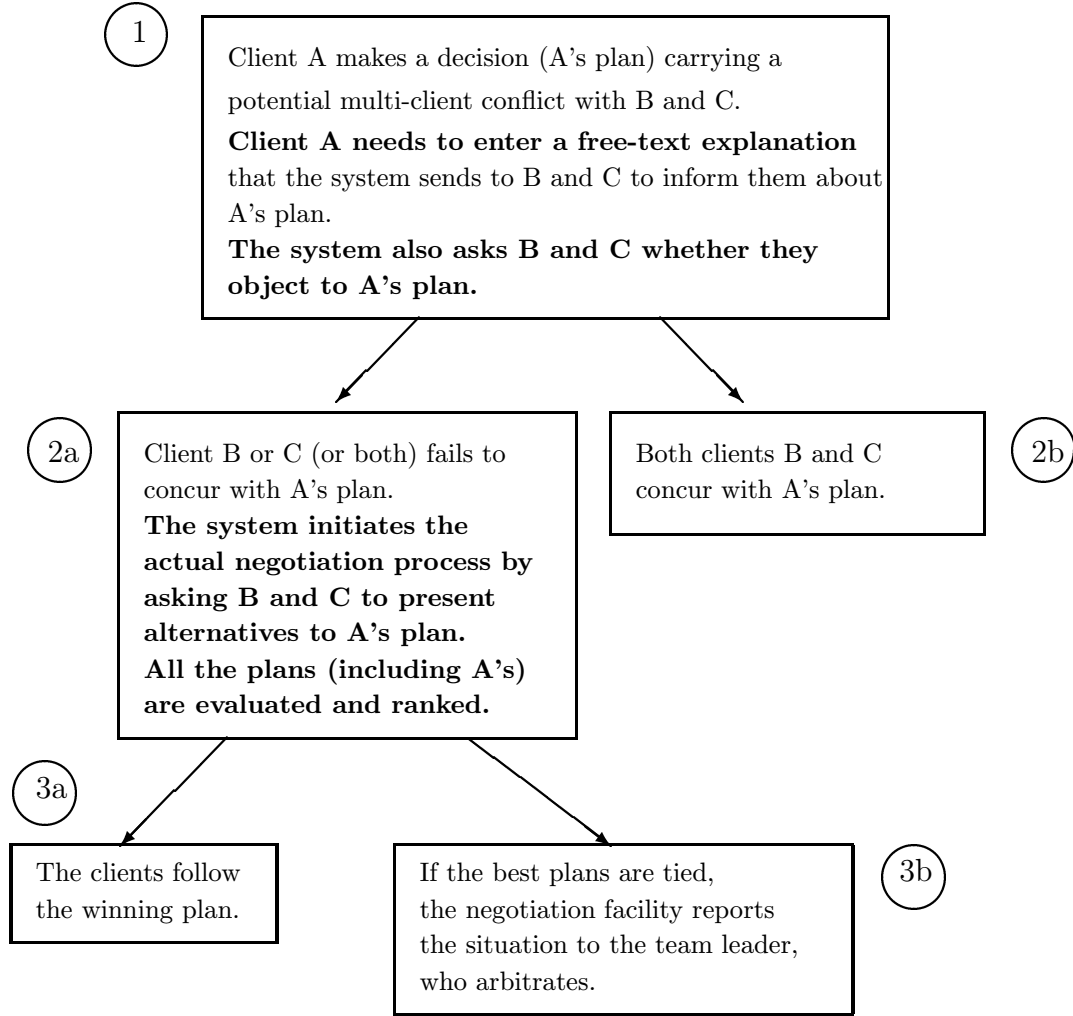


Figure 2: Negotiation process with the new protocol

---

these properties. Specifically, in our protocol, clients are required to define issues, which are the properties that the plans must satisfy. We then give procedures to compute the degrees of effect of the plans and their overall scores.

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 the **prioritization of system suggestions**, involving a single client.

## 6.2 Evaluation of plans

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 new decision. 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 modifiable and 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<sup>3</sup>, 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, a plan) that causes a potential multi-client conflict between 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) decides 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 preclude this type of thrashing. Fig. 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. To relate the scoring scheme to the theory of utility mentioned earlier, we try to keep the utility-space position of every agent as far from the origin as possible, but we do not restrict plans solely to moves away from the origin.

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.

In 1906, the economist V. Pareto introduced a notion of allocative efficiency, now more often called **Pareto optimality**: “a situation in which no reorganization or trade could raise the utility or satisfaction of one individual without lowering the utility or satisfaction of another individual” [14]. In other words, no one can be made better off without making someone else worse off. The Plan Evaluator selects a Pareto optimal solution plan among all feasible solutions. At any time, however, a client may make a decision that has the effect of rendering a formerly feasible solution infeasible because it violates a newly-introduced constraint. When this happens, the Plan Evaluator must select a new, Pareto-optimal solution from the new feasible set.

### 6.2.1 Determination of the affected issues

All client decisions, as well as the application of a plan, potentially result in changing the value of one or more parameters. Whatever the source of these 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 change in 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 for another parameter  $p_5$ . Then the parameter changes are:

- $p_2$  was assigned a new value  $v'_2$ , and
- member  $p_2$  of structured domain  $p_5$  was assigned a new value  $v'_2$ .

Therefore, the affected issues are any that contain  $p_2$  or  $p_5$ .

### 6.2.2 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 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.

---

<sup>3</sup>Remember that in a Galileo4-style system, *all* interaction consists of adding or retracting constraints.

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  $\Delta j$  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.

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 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.

### 6.2.3 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  $P$ :

$$\min_i \sum_{j \in J_i} (\alpha_{ij} \times f_{ij}) \quad (1)$$

where  $i$  represents each affected issue,  $J_i$  is the set of all parameters of issue  $i$ , and  $j$  represents each parameter in each issue with its weight  $\alpha_{ij}$  and partial degree of effect  $f_{ij}$ .

$\sum_{j \in J_i} (\alpha_{ij} \times f_{ij})$  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 all of them and recommends for adoption the plan with the highest score. See Scenario 1 in Sec. 7 for an example.

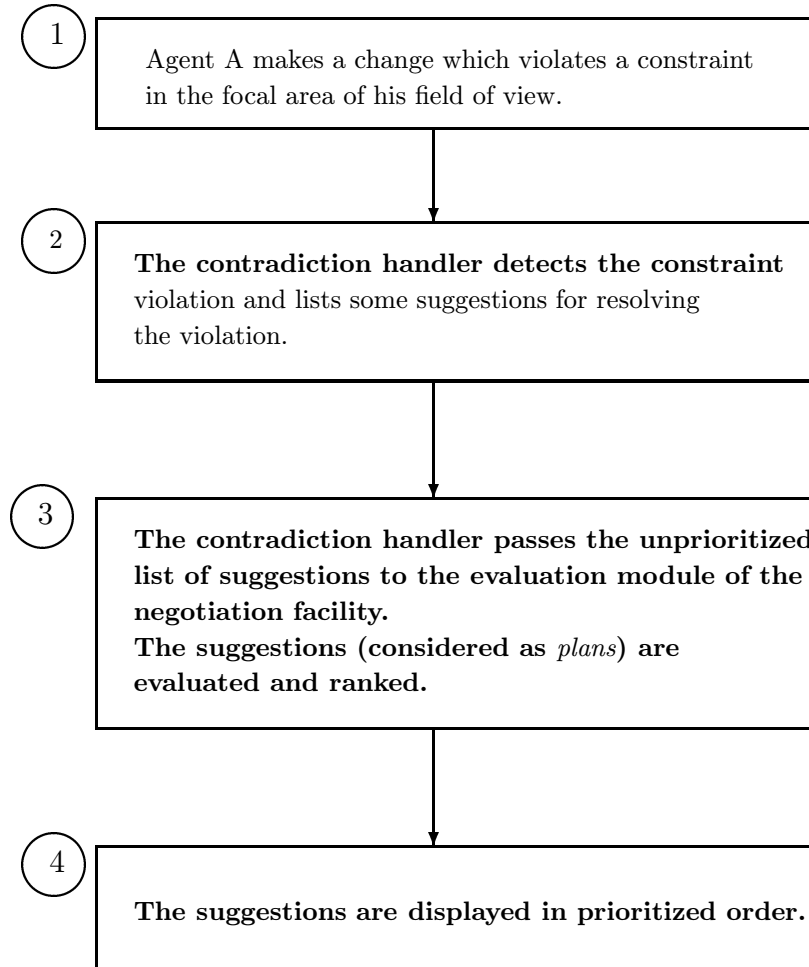


Figure 3: Prioritization of system suggestions using the new protocol

---

### 6.3 Approximated evaluation

In addition to plans offered by clients, the Plan Evaluator can also handle system or client suggestions that are too vague to be well-defined plans, e.g., suggestions to increase or decrease the value of a parameter without specifying to which new value. In these situations we cannot use the formulae of partial degrees of effect given above, so we approximate the evaluation of these suggestions by using just the weights  $\alpha_{ij}$  and the value of  $sgn_{ij}$  for all the parameters of an affected issue. See Scenario 2 in Sec. 7 for an example.

### 6.4 A third use for plans and issues

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. In Galileo terms, 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 Fig. 3. See Scenario 3 in Sec. 7 for an example.

## 7 Example scenarios

Suppose we have a design team involved in the development of a new circuit board, and that three perspectives are represented: configuration, production, and test. Suppose the configuration designers set forth this issue:

```
issue 'functionality'(func) for configuration ::=
{ min(no_components) * 0.6 using decr_duf,
  min(pwr_consump) * 0.4 using incr_duf }.
```

Suppose the production engineers have this issue:

```
issue 'manufacturability'(manuf) for production ::=
{ min(no_hole_sizes) * 0.2 using const_duf,
  min(time_lost_switching_machines) * 0.4 using const_duf,
  max(eqpt_utilization) * 0.2 using decr_duf,
  min(prod_eqpt_price) * 0.2 using decr_duf }.
```

Finally, suppose the test engineers have this issue (in which the default weights are used):

```
issue 'testability'(test) for test ::=
{ min(test_eqpt_price) using decr_duf,
  min(no_components) using const_duf }.
```

To save space in the tables that follow, we denote the first parameter of issue **func** (**no\_components**) by **func.1**, the second (**pwr\_consump**) by **func.2**, and similarly for all issues and parameters.

### 7.1 Scenario 1: potential multi-client conflict

In this scenario, designers will make a decision causing a potential multi-client conflict with the production engineers, so that negotiation is needed. Suppose the designers want to modify the set of components on a circuit board from A, B, and C to some components A and D in order to reduce the number of components (minimize the parameter **no\_components**). Suppose the components A, B, and C use the same hole size **size1**, whereas the component D uses a different hole size, **size2**.

Therefore, the number of required hole sizes would increase from 1 to 2 with the designers' decision, and this change would negatively affect the parameter **no\_hole\_sizes** 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 components by replacing the components B and C by the component D. The system informs the production engineers about the designers' intention of replacing the components B and C by the component D, 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 hole sizes 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 components B and C by the component E rather than by the component D, because the component E uses the same hole size (**size1**) as the component A. Suppose component E, however, would increase power consumption from 100 to 200 units.

The designers' decision constitutes Plan 1. The production engineers' counter-suggestion constitutes Plan 2. Table 1 summarizes the computation of the overall score of Plan 1 and Plan 2.

Plan 1 affects two issues: **func** and **manuf**. Within the issue **func** only one parameter is affected: **no\_components** (**func.1**), so the degree of effect of the issue **func** simply equals the product of the weight of parameter **func.1** by its partial degree of effect.

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

	Affected Parameters	$\Delta j$	Domain	Desired dir. of change	$sgn_{ij}$	Type of d.u.f.	$f_{ij}$	$\alpha_{ij}$	$\alpha_{ij} \times f_{ij}$
Plan 1	func.1	-1	[0,100]	min	+1	decr_duf	+0.33	0.6	+0.2
	manuf.1	+1	[0,20]	min	-1	const_duf	-0.05	0.2	-0.01
Overall score for Plan 1	$\min(+0.2, -0.01) = -0.01$								
Plan 2	func.1	-1	[0,100]	min	+1	decr_duf	+0.33	0.6	+0.2
	func.2	+100	[0,500]	min	-1	incr_duf	-0.25	0.4	-0.1
Overall score for Plan 2	$+0.2 + (-0.1) = +0.1$								
Output	Plan 2 selected								

Table 1: Scenario 1 – Evaluation of plans from multiple clients

which here is equivalent to:

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

. Thus, since  $\alpha_{ij} = 0.6$ , the degree of effect of Plan 1 on the issue `func` rounds to +0.2.

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

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

which here is equivalent to:

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

. Thus, since  $\alpha_{ij} = 0.2$ , the degree of effect of the issue `manuf` rounds to -0.01, and the overall score for Plan 1 is  $\min(+0.2, -0.01) = -0.01$ .

Plan 2 affects one issue, `func`, but within this issue two parameters are affected: `no_components` (`func.1`) and `pwr_consump` (`func.2`). Thus the score for Plan 2 equals the sum of two terms: (i) the degree of effect through `func.1` (+0.2), which is the product of the weight of `func.1` and its partial degree of effect due to Plan 2, and (ii) the degree of effect through `func.2` (-0.1), which is the product of the weight of `func.2` and its partial degree of effect due to this plan. The score for Plan 2 rounds to +0.1.

Plan 2 has a higher score than Plan 1 (+0.1 vs. -0.01). Therefore it is selected by the Plan Evaluator as the best plan, and the designers are asked by the system to replace the components B and C by the component E.

## 7.2 Scenario 2: approximate evaluation

The second scenario illustrates how evaluation can be approximated even for suggestions that are too vague to be well-defined plans, such as suggestions for increasing or decreasing the value of a parameter without specifying to which new value. For system or client suggestions that are too vague to be defined as a plan, we cannot use the formulae of partial degrees of effect given above. So we approximate the evaluation of these suggestions by just using the weights  $\alpha_{ij}$  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, and therefore at least one compromise suggestion violates the definition of a plan and the partial degrees of effect cannot be computed. Here are the assumed effects of both suggestions:

- Suggestion 1 would affect three parameters:
  - `no_components` (`func.1`), whose value would increase by 3,

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

Table 2: Scenario 2 – Approximate evaluation of non-plan suggestions

- `no_hole_sizes` (`manuf.1`), whose value would increase by 1, and
- `time_lost_switching_machines` (`manuf.2`), whose increase is not quantified.
- Suggestion 2 would affect an overlapping set of three parameters:
  - `no_components` (`func.1`), whose value would decrease by 1,
  - `no_hole_sizes` (`manuf.1`), whose value would increase by 2, and
  - `eqpt_utilization` (`manuf.3`), whose value would increase by 10.

Suggestion 2 is a well-defined plan because all its suggested changes are fully specified. But, since Suggestion 1 does not specify how much the client should increase parameter `time_lost_switching_machines`, Suggestion 1 cannot be defined as a plan, and the partial degree of effect for that parameter cannot be computed using the method presented earlier.

However, the degree of effect for each issue can be approximated by:

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

As shown in Table 2, Suggestion 2 wins, since its score is 0 versus  $-0.6$  for Suggestion 1.

Notice that the difference between this and the earlier formula (Eq. 1) is that  $f_{ij}$  has been replaced by  $sgn_{ij}$ . 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.

### 7.3 Scenario 3: prioritization of single-client alternatives

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 thus change the value of parameter `no_hole_sizes` (`func.1`) from 2 to 7 (call this Plan 1), or retract constraint  $C_b$  and thus change the value of parameter `time_lost_switching_machines` (`manuf.2`) 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 Table 3. 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	$\Delta_j$	Domain	Desired dir. of change	$sgn_{ij}$	Type of d.u.f.	$f_{ij}$	$\alpha_{ij}$	$\alpha_{ij} \times f_{ij}$
Plan 1	func.1	+5	[0,100]	min	-1	const_duf	-0.05	0.2	-0.01
Overall score for Plan 1	-0.01								
Plan 2	manuf.2	-10	[0,100]	min	+1	const_duf	+0.1	0.4	+0.04
Overall score for Plan 2	+0.04								
Prioritized suggestions	1. Select Plan 2 2. Select Plan 1								

Table 3: Scenario 3 – Prioritization of system suggestions

## 8 Comparison with Other Work

The Designer Fabricator Interpreter (DFI) [17, 18] 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 [15], a negotiation system for labor management conflicts (a non-cooperative environment), is one example of a system that uses case-based reasoning to resolve multi-agent conflicts. The drawback of this method is that when the current conflict does not match any past case stored in the memory, the system cannot provide a compromise solution unless it has available a second method capable of generating a solution from scratch. PERSUADER 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) [6]. Another aspect of this project is SHADE [11, 16], 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' [12] 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 [9] 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.

## 9 Conclusions

### 9.1 Contributions

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 evaluation procedure can be applied to the suggestions of clients whether they follow the current definition of a plan or not. Knowing only the direction of change of the affected parameters is sufficient to approximate the degree of effect of the plan for the issues containing the affected parameters.

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.

### 9.2 Current limitations and future research

Our protocol is currently limited to parameters defined over totally ordered bounded domains. Some extensions would be needed to allow the computation of the degrees of effect of plans on issues containing parameters defined over unbounded domains and/or partially ordered bounded domains. The underlying language could be extended to allow clients to define their own partial orderings and refer to them in the issue definitions for a more general meaning of the keywords `min` and `max`. The definition of an issue could also be generalized by allowing any expressions (such as  $5 \times X - 9$ ) instead of only parameters (such as  $X$ ).

The current definition of a plan is restricted to the assignment of values to parameters, and allows only adding a linear equality constraint, or retracting any constraint that was underlying a contradiction. For full functionality plans should be extended by allowing:

- The retraction of a constraint where that has the effect of making one or more parameters unknown. Assume that a parameter  $X$  appears in a single constraint  $X = Y - 1$  and that  $Y$  has the value 4.  $X$  then has the value 3. If the constraint is retracted, the parameter  $X$  no longer has a known value.
- The addition of a constraint expressing an inequality, like  $X < 0.1$ , which does not force the parameter  $X$  to a single value.
- Similarly, the addition of a non-linear equation, like  $X^2 - 3 \times X + 2 = 0$ , which restricts the set of possible values of  $X$  only to the set  $\{1, 2\}$ , rather than to a single value.
- The introduction and use of new system-generated parameters, since any language that adheres to the Galileo4 model must allow this conditional existence of parameters.
- Adjustments to the priority of a constraint by increasing or decreasing its hardness value.

Among other areas of future work are exploring the effect on plans of new system-inferred parameters such as are possible with Free Logic constraints, and support for multiple exploratory designs-in-progress developed concurrently by the same participants.

## References

- [1] Bahler, D., C. DuPont, and J. Bowen, "Negotiation in client-assisted inference systems for Concurrent Engineering," *Workshop on Computational Models of Conflict Management in Cooperative Problem Solving*, Chambery, France, August 1993, 23-40.
- [2] Bowen, J. and Bahler, D., "Conditional Existence of Variables in Generalized Constraint Networks," *Proc. of 9th Nat. Conf. on AI (AAAI-91)*, Anaheim, 1991, 215-220.

- [3] Bowen, J. and Bahler, D., "Supporting Cooperation between Multiple Perspectives in a Constraint-based approach to Concurrent Engineering," *Journal of Design & Manufacturing*, (1), 1991, 89-105.
- [4] Bowen, J. and Bahler, D., "Frames, quantification, perspectives and negotiation in constraint networks for life-cycle engineering," *Int. Journal of AI in Engineering*, (7), 1992, 199-226.
- [5] Bowen, J. and Bahler, D., "Constraint-Based Software for Concurrent Engineering," *IEEE Computer*, Special Issue on Computer Support for Concurrent Engineering, 26(1), January 1993.
- [6] Cutkosky M., Engelmores R., Fikes R., Gruber T., Genesereth M., Mark W., Tenenbaum J., and Weber J., "PACT: An Experiment in Integrating Concurrent Engineering Systems," *IEEE Computer*, Special Issue on Computer Support for Concurrent Engineering, 26(1), January 1993.
- [7] Dupont, C. and Bahler, D., "A Negotiation Protocol for Client-Assisted Inference Systems in Concurrent Engineering," Technical Report, North Carolina State University, 1993.
- [8] Erman, L.D., Lark, J.S., Hayes-Roth, F., "ABE: An Environment for Engineering Intelligent Systems," *IEEE Transactions on Software Engineering*, SE-14, 1988, 1758-1769.
- [9] Garcia, A.C.B. and H.C. Howard, "ADD's Model for Acquiring Design Rationale," *Proc. AAAI '92 Workshop on Design Rationale Capture and Use*, San Jose, CA, July 1992, 91-98.
- [10] Gasser, L., Braganza, C., Herman, N., "MACE: A Flexible Testbed for Distributed AI Research," in M.N. Huhns (ed.), *Distributed Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann, 1987, 119-149.
- [11] Gruber, T.R., Tenenbaum, J.M., and Weber, J.C., "Toward a Knowledge Medium for Collaborative Product Development," in J.S. Gero (ed.), *Artificial Intelligence in Design '92*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 1992, 413-432.
- [12] Petrie, C., "The REDUX' Server," *Proc. Intl. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, May 1993.
- [13] Rosenschein, J.S., Genesereth, M.R., "Deals among Rational Agents," *Proc. 9th Int. Joint Conf. on AI (IJCAI-85)*, Los Angeles, 1985, 91-99.
- [14] Samuelson P.A., Nordhaus W.D., *Economics*, 14th ed., New York: McGraw Hill, 1992.
- [15] Sycara, K.P., "Multiagent compromise via negotiation," in L. Gasser and M. N. Huhns, eds., *Distributed Artificial Intelligence 2*, London: Pitman/Morgan Kaufmann, 1989, 119-138.
- [16] Tenenbaum, J., Weber, J., and Gruber, T., "Enterprise Integration: Lessons from SHADE and PACT," in C. Petrie (ed.), *Enterprise Integration Modeling*, Cambridge, MA: MIT Press, 1992.
- [17] Werkman, K.J., "Using negotiation as a means of coordinating distributed problem solving," *Proc. World Congress on Expert Systems*, Orlando, FL, December 1991.
- [18] Werkman K.J., "Cooperative design evaluation between multiple agents using negotiation with shareable perspectives," IBM Corporation monograph, 1992.