Combining Constraint Propagation and Backtracking for Distributed Engineering

Charles Petrie
Heecheol Jeon
Mark R. Cutkosky
Center for Design Research
Stanford University
560 Panama Street
Stanford, CA 94305-2232
petrie@cdr.stanford.edu

Abstract

We describe an agent-based approach for engineering problems in which the constraints and general control of problem solving are distributed. In order to overcome previous problems with engineering constraint solvers, we divide responsibilities between domain-specific agents, which control the overall problem solving, and generic, reusable agents. One of these generic agents is *Redux′*, which provides general problem solving bookkeeping services. Another is a *Constraint Manager*, which provides constraint consistency services. We demonstrate the utility of this approach on a previously-defined simple, but difficult, distributed constraint problem.

This work was funded by Navy contract SHARE N00014-92-J-1833 under the US DARPA RaDEO program.

An Engineering Constraint Manager Agent

Constraint solving is a common function in engineering projects. As distributed and concurrent engineering become more common, the requirement to solve various kinds of constraints spanning disciplines becomes more important. The difficulty of this requirement is increased when the participants are not co-located and when constraint solving must be interleaved with general design and project management.

We are exploring one obvious approach: KQML-based agents. With this technology, one can "wrap" legacy tools (e.g., CAD systems) with software, usually called Application Program Interfaces (APIs), that allow them to communicate via a common agent protocol, in this case, KQML(Finin et al. 1992). We are attempting to define reusable, generic KQML-agents useful for engineering design. We call the whole framework of reusable messages and agents *Process-Link*[1]

The core generic agent is *Redux′*(Petrie 1993), which assumes a particular model of the design process in which design decisions are to divide an issue or a task into subissues or subtasks, and/or to make statements about the design. Thus we say a *decision* consists minimally of a *goal* (the issue or task), and *results*, consisting of at least one *subgoal* or at least one *assignment* (a statement about the design). *Constraints* may be violated by some set of assignments. Constraint violations are resolved by problem solvers rejecting/changing decisions that resulted in conflicting assignments.

This model is consistent with the generate-and-test model of design. A decision is made, assignments generated, and a test made to see if the assignment is consistent with the set of known active constraints. If not, the user may choose to leave the design inconsistent, or, at any time, may reject one or more of the decisions that led to the inconsistency. One important function of *Redux′* is to remember that the rationale for one decision might be the rejection of another, and, in turn, the reasons for the rejection. This detail has been documented elsewhere(Petrie et. al 1995) and is outside the scope of this paper, but the basic functionality is dependency-directed backtracking (DDB)(Stallman and Sussman 1976).

Generate-and-test with DDB allows for incremental rejection of design decisions, and thus the maintenance of rejection reasons useful for design. But generate-and-test is much less efficient than constraint propagation for many engineering problems. Thus, some constraint solver should be combined with *Redux′*. A major challenge is to be able to combine DDB with constraint propagation and generate rejection reasons as needed.

Another issue is that there are many different kinds of constraints; e.g., algebraic with continuous do-

[1] This is documented at *http://cdr.stanford.edu/ProcessLink/* on the WWW.

mains, symbolic with discrete domains, etc. And there are many different extant constraint satisfaction systems. Our goal, therefore, was to construct a generic *Constraint Manager (CM)* that would allow different solvers to "plug and play", route constraint solving requests to appropriate solvers, and send not only consistency information, but also rejection reasons as needed to requesters. And the messages exchanged between the CM,*Redux′*, solvers, and requesters should be domain-independent.

We also noted that the basic constraint formalism of variables is not sufficiently structured for engineers who want to reason about which type of object to use in an assembly based upon its features and problem requirements. Thus the CM had to not only translate among solvers but also to include a better engineering formalism.

While we have made substantial progress on these last two issues with the CM, they are mentioned only for context as this paper reports only on the fundamental issues of control of problem solving and combining constraint propagation and DDB.

Control of Constraint Solving Issues

In developing the CM for engineering applications, we have addressed the following problems:

- *Constraints and variable domains change* during problem solving. A constraint or variable may be the result of a specific design decision, which may later be rejected in favor of another decision. This revision is unlimited. For example, as engineers decide to use a part, new variables corresponding to its features come into play. If the part is discarded (perhaps only temporarily) from the design, so are those variables and associated constraints.

- Especially in collective problem solving, *incremental reasoning* is important in engineering design. If the constraints or domains change, it is inefficient to start from scratch and resolve the whole problem each time. Perhaps more important to the user, the resulting answers may be very different from the current context of problem solving(Dhar and Raganathan 1990).

- Engineers want *control over problem solving*. They don't want a constraint satisfaction system to give them the answer; they want information about consistent alternatives(Park et al. 1994). They may even prefer to live with some inconsistencies for a period of problem solving.

- Since most constraint propagation algorithms do not ensure global consistency, *backtracking is necessary.* This is also true for overconstrained problems and for many situations where generate and test is an appropriate problem solving strategy. In these cases, engineers want to know the constraints, domains, and decisions that might be changed to remove the inconsistency(Park et al. 1994).

- In distributed collective design, *changes in constraints, variables, domains, and assigned values have various effects* on design decisions by different engineers. These effects should be noted.

Task Decomposition

Many have noted that constraint propagation should be used as a filter for a distinct preprocessing task. Dechter and Pearl(Dechter and Pearl 1989) and Mackworth (Mackworth 1988) all refer to constraint propagation as "preprocessing". One generic model of configuration design problem solving refers to ruling out infeasible boolean expressions prior to selection of design extensions(Balkany et al. 1993). Gaertner and Mitsch(Gaertner and Miksch 1995) identify those points in scheduling at which the problem solver should intervene to improve the tractability of constraint satisfaction. This is in contrast to approaches that approach design problem solving completely as a constraint solving task(Bowen and Bahler 1992).

Separating *problem solving* from constraint propagation allows more control over problem solving. We extend this notion to say that there is a general service of *consistency management* that includes constraint propagation before decision making and consistency checking afterwards[2].

We extend further this decomposition in identifying a separate task of dependency-directed backtracking*(DDB) bookkeeping* that is especially important for incremental revision by multiple problem solvers with dynamic constraints and domains. In order to determine the incremental effects of changing conditions, more DDB support is required than is provided by strictly constraint-based approaches such as Dechter's dynamic constraints (Dechter and Dechter 1988) or even the backtracking of (Mittal and Falkenhainer 1990).

The problem solver can not only make choices of values to be assigned to variables, but can also reject these assignments at will. This may occur in response to a constraint violation. DDB support provides the problem solver, upon request, with the AND/OR tree of assignments responsible for the conflict. Given a set

[2]Also included is the matching of constraint types to constraint solvers and the translation of constraints and results among them.

of assignments to reject, the reasons for the rejection should be noted in order to 1) avoid thrashing and 2) determine when the rejection is no longer necessary.

This bookkeeping is especially desirable if constraints and variable domains can change. Part of this task is to determine the effect of these changes on the current solution and inform the affected agents. For instance, the deletion of a constraint may mean that a known conflict is no longer the case and an earlier preferable solution may be possible(Petrie et. al 1995). Not only variable assignment values, constraints, and domains, but also choices about goal composition may be revised in response to constraint violations. The DDB bookkeeping must be done in all cases. In particular, we say that a goal, variable value assignment, constraint, variable domain definition, or rejection is *valid* or *invalid* depending upon conditions determined by the problem solver[3]

Consideration of the DDB bookkeeping task suggests more work for the consistency management task. Constraint propagation and DDB are dissimilar constraint satisfaction techniques. The former filters out some inconsistent values prior to a choice. The latter is used when further inconsistencies necessitate rejection of choices. But overconstrained problems are common. In these cases, DDB needs to consider constraints and choice of domains as well as assignments used to filter choices during constraint propagation. Then consistency management must perform some K-consistency algorithm(Freuder 1978) to generate rejection reasons that include these constraints and domains for use by the DDB bookkeeping function.

Agent Responsibilities and Interactions.

Agent technology provides a good structure for separating tasks and making generic functionality easily available. The constraint management task divisions of Section suggest agent responsibilities and interactions. The problem solving is performed by domain-specific agents, DDB bookkeeping by *Redux'*, and consistency management by the Constraint Manager, as illustrated in Figure 1, which also suggest some of the messages that must be passed among such agents.

Definition of the messages to be exchanged and the functional responsibilities of the agents is a major technical challenge. We describe here the overall principles omitting implementation. A full description of the KQML messages that the current implementation exchanges is beyond the scope of this paper and may only be one way of implementing this functionality. The

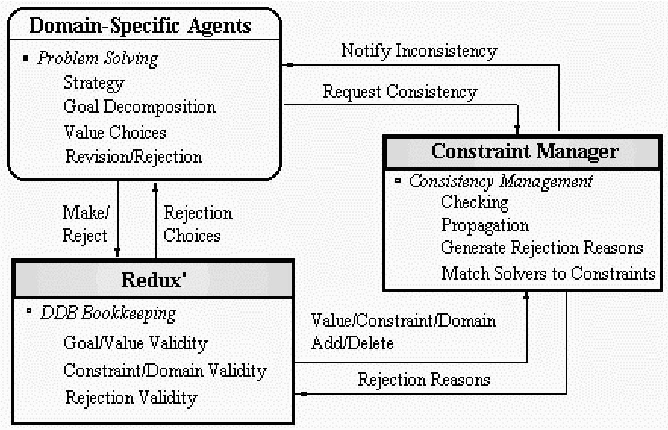

Figure 1: Constraint Solving Tasks and Agents

types of messages exchanged between a problem solver and *Redux'* are covered in (Petrie et. al 1995). The basic messages from the problem solver to the CM are requests for elimination of variable domain choices via constraint propagation. The CM will return the consistent vs inconsistent choices prior to decision making, using constraint propagation. The CM also performs automatic consistency checking as variable value assignments are made by decisions. The CM remembers consistency requests and replies and will later initiate a message if changes, such as a domain change or an assignment being made, would cause a different response, including an overconstrained condition in which no choices are any longer consistent[4].

For interaction with the CM, we allow variable domain definitions to be a special case of *assignments*. Thus, a decision might result in the assignment `Domain-Definition (section-modulus beam-1) [3-15]`, which may be changed later in problem solving.

Addition or relaxation of a constraint, a change in a domain definition, or the addition or invalidity of an assignment may mean that a reply by the CM to an earlier consistency question needs revision. *Redux'* will always inform the CM of changes in variable domain and constraint definitions or addition/deletion. The CM keeps a record of variable consistency requests and replies for the duration of a project. If the current change involves one of these variables, the CM will recompute consistency and inform the requesting agent if the new consistent domain is different from the old one. The domain-specific agent may in turn decide to revise its design decision, causing further incremental

[3] *Validity* in the Redux model is analogous to the classical logical sense of being a valid consequence, usually denoted by the entailment symbol $\models$.

[4] At "http://cdr.stanford.edu/ProcessLink/", see "protocol/EPL-syntax.html" and "ConstraintManager/" for details.

changes to the design state.

A special case involves the "activation" of constraints by the CM. If a variable domain definition is removed by the rejection of a decision, and not replaced by a new one, then the variable is not currently important for problem solving. For instance, if there is no valid assignment of a value to the variable `(section-modulus.beam-1)` and no valid domain definition for this variable, then *Redux′* notifies the CM, which omits constraints with this variable from further consistency checking or propagation, even if the constraint is valid.

The problem solver may also request rejection reasons. Suppose that the CM tells an agent that no values in the domain of a variable are consistent. The agent would then like to know why. Doing so requires that the *Redux′* agent and the CM work together to generate rejection reasons sufficient for control of problem solving[5].

In the case where all possible variable values are ruled out by constraint propagation, upon request, the CM will use a K-consistency algorithm(Freuder 1978) to generate the variable assignments and domains and constraints that directly caused the inconsistency[6]. The CM furnishes this information to the *Redux′* agent, which then determines which decisions by which agents were responsible for these assignments, domains, and constraints[7]. Obviously, this is an expensive computation and is the tradeoff to performing tractable but fallible constraint propagation.

The Secretaries' Nightmare

While the preceding section outlined the agent interactions, a scenario will illustrate in more detail the kind of messages that should be exchanged to provide the stated functionality.

In 1993, one of us, Petrie, organized a workshop in which researchers presented papers on how their approaches would relate to a simple scheduling problem, called "The Secretaries' Nightmare"[8] No one paper solved the whole problem, nor was any expected to do so. However, some aspects of the problem were particularly problematic and can be used to illustrate how the CM and *Redux′* work with problem solving agents within the Process-Link framework.

The problem can be summarized as follows:

Constraints: all-day meetings on weekdays in April; first meeting includes Axel, Brigitt, Dirk; second meeting includes Axel, Brigitt, Carl; if these two meetings are not held back-to-back, a third meeting, in between these two, must be held between Carl and Dirk; Carl is not to attend the first meeting.

Preferences: Back-to-back meetings and earlier dates.

Conditions: No central organizer, each person is free to change published availability dates as convenient. As examples, Dirk and Carl can change the third meeting constraint; anyone can propose or reject dates.

Domains: Axel is available in April the week of the 4th, the 18th and 19th, and the 25th and 26th. Brigitt is available the 7th, 8th, 19th, and the week of the 25th. Carl is available on the 7th, 19th and 26th. Dirk is available on the 7th, 8th, 18th, and 25th.

Modeling: Our method does not take into account how agents come to agree upon variable names, goals, etc. We assume that the agents propose common variables, domains, and constraints. Let Axel's meetings be *A-M1, A-M2, A-M3*, with similar notation for the others. The constraints are sent to *Redux′* by any one of the participant agents in *Constraint-Def* statements that include a constraint name, expression, and list of variables. We list the constraints here in the form `<name>:  <expression>` :

```
C-1:  (> Date.M1 Date.M2),
C-2:  (= Date.A-M1 Date.B-M1 Date.D-M1
Date.M1),
C-3:  (= Date.A-M2 Date.B-M2 Date.C-M2
Date.M2),
C-4:  (= Date.C-M3 Date.D-M3 Date.M3),
C-5:  (AND (> Date.M2 Date.M3) (> Date.M3
Date.M1))
C-6:  (= Date.M2 (+ 1 Date.M1))
```

There are various ways the agents could model their *Redux′* decisions. One we will use will be to make the scheduling strategies explicit. So, we say there is a top-level (root) goal of, say, *"Assign Meeting Dates for M-1 and M-2"*, called *Schedule-Meetings*. There

[5] *Redux′* actually does more than is what is commonly referred to as "dependency-directed backtracking" in many systems, such as those analyzed in (Balkany et al. 1993). The latter only identify the "nogoods" and allow the search to be recontinued at any one of the choices responsible for the conflict, without affecting the others. *Redux′* additionally tracks the reasons for the conflict and ensures that the search will not thrash, as noted in (Petrie et. al 1995).

[6] This idea was first suggested by Juergen Paulokat in 1995 in work on his dissertation.

[7] An AND/OR tree of resolution possibilities is generated since more than one decision can result in the same variable assignment.

[8] This was the Workshop on Distributed Scheduling at the 1993 IEEE CAIA. The original CFP, including the full description of the scheduling problem, and the presented papers are available at http://cdr.stanford.edu/html/people/petrie/caia.html.

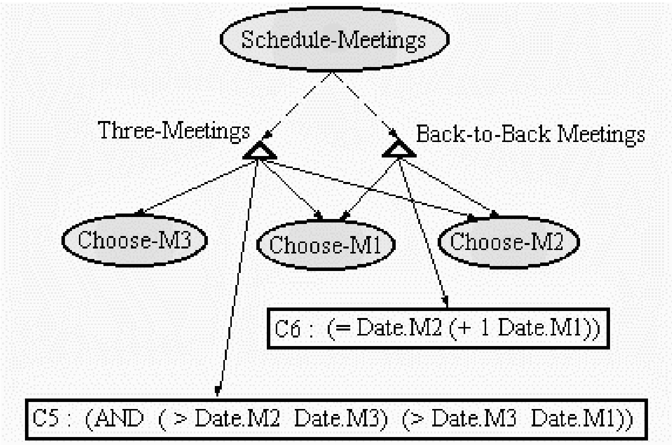

Figure 2: Two Possible Initial Decisions

are two possible decisions to be made, according to the preferences. One is to schedule back-to-back and the other is to schedule three meetings. Let us call the first decision *Back-to-Back* and the second *Three-Meetings*. Both decisions have goal *Schedule-Meetings* and, as results, subgoals *Choose-M1: Choose Date for M1* and *Choose-M2: Choose Date for M2*. However, these decisions differ in that *Three-Meetings* posts a third subgoal of *Choose-M3: Choose Date for M3*. We also extend the Redux model by allowing a new constraint to depend for its validity on a decision. In this case, were the decision *Back-to-Back* to be made, *Redux'* would inform the CM of the constraint `C6: (= Date.M2 (+ 1 Date.M1))`. The *Three-Meetings* decision results in the constraint `C-5: (AND (> Date.M2 Date.M3) (> Date.M3 Date.M1))`. These two possible decisions for goal *Schedule-Meetings* are illustrated in Figure 2 with goals as ovals and decisions as triangles.

The problem solving scenario can be organized in steps. For each step, we explain how the Process-Link agents would interact. Each of the meeting participants is an agent.

Step 0: Each agent publishes a domain definition; e.g. *Domain-Def Date.A-M1* {*4,5,6,7,8,18,25,26*} and *Domain-Def Date.A-M2* {*4,5,6,7,8,18,25,26*}.

Issue: Like constraint definitions, domain definitions are first sent to *Redux'*. *Redux'* always forwards constraint and domain definitions to the CM because these definitions may depend upon design decisions. Agents make changes to constraints and domains either directly, or indirectly through decision revision, and *Redux'* informs the CM of such changes.

Step 1: Upon requests by Axel, Bridget, and Dirk, the CM indicates two consistent solutions for the two meetings: [7, 19] and [25, 26]. Axel proposes that the earlier preference overrides the back-to-back meeting preference, makes decision *Three-Meetings*, and goes on to make decisions for *Choose-M1* and *Choose-M2* resulting in the assignments **Date.M1 := 7** and **Date.M2 := 19**, respectively. *Redux'* communicates these assignments to the CM. All the agents have registered an interest in such assignments, so they are notified as well[9].

Step 2: The requirement for a third meeting to be scheduled is only now introduced as the goal *Choose-M3*. The addition of C-5 causes the CM to request domain definitions for **Date.M3**. Carl and Dirk send these. Immediately, the CM detects that the problem is now overconstrained. The CM determines that the reasons are constraint *C-5*, the assignments **Date.M1 := 7** and **Date.M2 := 19**, and the domains for **Date.C-M3** and **Date.D-M3**.

Issue: The issue here is the "activation" of constraint *C-5*. In order to simulate backtracking functionality as well as to illustrate control of constraint solving, this constraint should not to be considered initially. We choose to make the constraint depend directly upon the decision*Back-to-Back*, rather than add conditions to the constraint. This has the advantage that as the *Three-Meetings* decision is made and rejected, constraint *C-5* comes and goes, and is considered by the CM only when appropriate. Constraint *C-6* is modeled similarly.

A notice of the conflict is sent by *Redux'* to all of the participants, as all of their domain definitions are implicated. Axel asks *Redux'* for the reasons and *Redux'*, based on information from the CM, includes the decision *Three-Meetings* as one of the causes. Axel responds by changing the decision to *Back-to-Back*, resulting in deactivation of *C-5* and activation of *C-6*. Variables **Date.C-M3** and **Date.D-M3** are no longer considered. After making a consistency request to the CM, Axel assigns **Date.M1 := 25** and **Date.M2 := 26**.

Step 3: However, now Dirk objects to the scheduled meeting as being too late and rejects **Date.M1 := 25**. The CM reports the problem is now *overconstrained.* All participants are notified as their domain definitions are implicated. Someone must expand their domain.

Issues: The problem solvers are free to reject solutions. If the problem becomes overconstrained, the CM and *Redux'* can state the reasons, including do-

[9] This is a *Redux'* feature called *RequestFeature.*

main definitions.

Step 4: Brigitt offers to cancel another meeting and be available on the 18th by sending a new domain definition. *Redux'* notices that the (overconstrained) conflict is potentially resolved and notifies the CM of the change. Remembering previous consistency requests, the CM notifies everyone that the two meetings can be held on the 18th and 19th. Dirk chooses to keep the *Back-to-Back* decision and remake the decisions about the choice of meeting dates for *M1* and *M2*, resulting in the assignments **Date.M1 := 18** and **Date.M2 := 19**.

Issues: The CM notification is based on a change to a previous response. Also, the context of scheduling back-to-back meetings is maintained.

Step 5: Then Carl's schedule changes so that he is available on the 8th. This change in his domain causes *Redux'* to send two messages: one to Axel suggesting that the *Three-Meetings* decision, together with [7,19], may be possible after all, and one to the CM, notifying it of the domain change. The CM sends Axel, Brigitt, and Dirk messages confirming that [7,19] is available because the third meeting can be scheduled on the 8th. Even better, the CM will note a back-to-back meeting on [7,8] is now possible. Brigitt, in order to make the 18th free again, changes the date decisions to result in the assignments **Date.M1 := 7** and **Date.M2 := 8**.

Issues: Redux' volunteers information about a possible change to an earlier context. The CM volunteers complementary information about variable value possibilities, based upon previous interest. Both mechanisms are necessary to note all the possibilities.

Step 6: Axel objects saying that he has already planned another meeting in Los Angeles on the 19th to take advantage of his presence there. He takes up the *Redux'* suggestion of *Three-Meetings*, reverts to this decision and to the decision for
Date.M1 := 7. (The decision for **Date.M2 := 19** is unchanged.) The goal to have a third meeting is reinstated and satisfied with a decision that **Date.M3 := 8**.

Issues: Problem solvers are free to use the volunteered information at any time. If it were no longer the case that this option was available, *Redux'* would not let the decision be made. In this case, as soon as the context changed, constraint C5 was in force, C6 was not, and the goal of choosing a value for M3 was again valid.

Problem Summary:

Step	*Decision*	*M1,M2,M3*	*Next Event*
1	Three-Meeting	7,19,x	C5 Added, Violated
2	Back-to-Back	25,26	Reject M1, Overconstrained
3	Back-to-Back	xx,26	C6 Violated, Domain Change
4	Back-to-Back	18,19	Domain Change
5	Back-to-Back	7,8	Reject M2
6	Three-Meeting	7,19,8	End

This problem shows that it is simple and yet difficult to support formally. Further, the difficulty of following even this simple example illustrates the need for a bookkeeping and notification service. The problem is representative of engineering projects in which domains and constraints change as components are added and deleted from the design and designers change their preferences. In particular, it illustrates control of problem solving strategy by the users, interleaved with constraint propagation, dependency-directed backtracking, and interactions among the agents.

Conclusions

The major principle discussed here is the separation of backtracking bookkeeping from consistency management from problem solving. Indeed, *separating consistency management from problem solving requires support for backtracking.* Problem solvers need to be able to work with inconsistencies and backtrack at will. DDB allows problem solvers to be *reminded of previous solutions* and what is now wrong or right with them. This preserves the history of collective problem solving, facilitating incremental change.

Conversely, *constraint propagation complements DDB* by eliminating the necessity to generate conflicts in order to detect the effects of constraint deletion.

The backtracking bookkeeping also should *prevent problem solving thrashing.* Though not illustrated, the problem solvers are free to pick inconsistent solutions and delay their resolution indefinitely. However, problem solvers are not allowed to rechoose a rejected decision while the reasons for the rejection are still valid. Moreover, these rejection reasons are used to warn problem solvers if they may be entering in a cycle of making and rejecting old decisions.

Further, *consistency calculations must be based on the current state of problem solving.* This means the CM needs to be notified by the bookkeeping agent whenever the validity of a constraint, domain, or value assignment changes, as well as of the addition of new ones. In turn, the CM needs to *remember previous* *consistency requests* by problem solvers, and *notify the requester later* if the answer would be different given the change of state detected by the bookkeeping agent.

What does the problem illustrate? First, notice that the representation of *C-5* can be very simple. It need not have any conditions about whether the other two meetings are back-to-back, as that is modeled in the decision itself. There is no need for a *exists* relation, which does not properly belong in constraints anyway. The constraint comes into play only when appropriate. While this condition could be modeled within the constraint itself, it imposes more work on both the constraint writer and the constraint solver, and such conditions may not be easy to express in more complicated engineering design problems.

Second, the need for backtracking is fundamental for hard (read practical) problems. The revision and notification illustrated could not be handled by any of the systems at the workshop from which this example was drawn and has caused problems in others.

We have presented a scenario that illustrates the following:

- management of constraint and variable domain change,
- simplification of constraint definitions, as with *C-5* and *C-6*,
- the CM and *Redux'* suggesting previous solutions,
- generation of the reasons for rejection of alternatives in an overconstrained state, and
- the problem solving agents freely choosing and rejecting solutions based on arbitrary preferences.

The implementation status is that we have a prototype CM and message protocol implemented that can solve the secretaries' nightmare problem in conjunction with *Redux'* within the Process-Link agent framework. The CM also works with the Next-Link electrical cable harness design application(Park et al. 1994) and we are developing other applications within the mechanical engineering domain while refining the agent message protocol, as well as the constraint notation and the ability to incorporate different types of constraint solvers.

References

Balkany, A., Birmingham, W.P., Tommelein, I.D., "An analysis of several configuration design systems,"*Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI/EDAM)* **7**:1, 1-18, 1993.

Bowen J. and Bahler D., "Task Coordination in Concurrent Engineering", *Enterprise Integration Modeling*, C. Petrie, ed., MIT Press, October, 1992.

Dhar V. and Raganathan N., "An Experiment in Integer Programming," *Communications of the ACM*, March 1990.

Dechter R. and Pearl J., "Tree clustring for Constraint Networks," *AI* **38** 353 - 366, 1989.

Dechter, R.J and Dechter, A., "Belief Maintenance in Dynamic Constraint Networks, *Proc. AAAI-88*, 37-42, St Paul MN, 1988.

Finin, T., Fritzson, R., & McKay, D., "A Language and Protocol to Support Intelligent Agent Interoperability," *Proc. of the CE & CALS '92 Conf.*, Washington , June 1992. See also *http://www.cs.umbc.edu/kqml/* on the WWW.

Freuder, E.C., "Synthesizing Constraint Expressions," *Communications of the ACM*, **21**:11, 958 - 965, 1978.

Gaertner, J. and Miksch, S., "Shift Scheduling with the Projections First Strategy," Oesterreichisches Forschungsinstitut für Artificial Intelligence - OeFAI, Technical Report, Vienna, 1995.

Mackworth, A., "Knowledge Structuring and Constraint Satisfaction:The Mapsee Approach," *IEEE PAMI*, Nov. 1988.

Mittal, S. and Falkenhainer, B., "Dynamic Constraint Satisfaction," *Proc. AAAI-90*, 25-32, MIT Press, 1990.

Park, H., Lee, S., and Cutkosky, M., Section 5 of "An Agent-Based Approach to Concurrent Cable Harness Design," *AI/EDAM*, **8**: 1, 1994.

Petrie, C., "The Redux' Server," *Proc. Internat. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, May, 1993.

Petrie, C., Webster, T., & Cutkosky, M., "Using Pareto Optimality to Coordinate Distributed Agents," *AI/EDAM*, **9**, 269-281, 1995.

Stallman, R. and Sussman, G., "Forward Reasoning and Dependency-Directed Backtracking," Memo 380, Massachusetts Institute of Technology, AI Lab., Sept. 1976.