DESIGN SPACE NAVIGATION AS A COLLABORATIVE AID

CHARLES PETRIE AND MARK CUTKOSKY
Center for Design Research
Stanford University
560 Panama Street
Stanford, CA 94305-2232
petrie@sunrise.stanford.edu

AND

HISUP PARK
Lockheed Missles and Space Co.
Artificial Intelligence Center
Palo Alto, CA
park@sunrise.stanford.edu

Proceedings of the Third International Conference on Artificial Intelligence in Design, Lausanne, August, 1994

Abstract

The *Redux'* server is an agent, with no domain-specific knowledge, that provides generic coordination services to distributed design systems. The coordination is accomplished by a "wrapper" technique requiring relatively little modification of existing agents. Yet the coordination services significantly extend the usual "ask/tell" agent protocols. These advantages are obtained because the server is based on a simple and ubiquitous search-based model of design decisions. The central hypothesis is that routine design can be considered search and design agents can be coordinated by management of the search space.

The *Redux'* services and the mapping to existing agents are illustrated here by describing how an existing distributed configuration design system, *First-Link*, is being developed into a new system, *Next-Link*, with additional functionality.

1. Introduction

The *First-Link* project in the Stanford Center for Design Research is sponsored by Lockheed[1] in order to develop a distributed system to design cable harnesses for aircraft (Park *et al.*, 1994), and as part of the SHARE (Toye *et al.* , 1993) project that is investigating computer support for distributed design. The design domain is typical of the generic configuration design task, but also tackles tough geometric modeling problems. A single platform version of this system is planned for commercial use next year. The prototype of the distributed design system has a simple agent interaction model that we are augmenting with *Redux′* (Petrie, 1993), an agent implementing a subset of the full Redux model described in (Petrie, 1992). This new version, *Next-Link*, will have more sophisticated agent interactions, as described here.

The Redux model is based upon AI problem solving notions of search spaces. Redux is a model of solving task/resource assignment problems with constraints and subgoaling. *Redux′* is a subset of the full model that treats only the maintenance of dependencies generated by any design problem solver.

The central idea of using *Redux′* as a coordination facilitator is that the domain-specific agents performing the distributed design depth-first search a shared design space. Each agent is exploring a part of that space. Any design decision made by any one agent can alter that space, affecting the positions of the other agents, just as masses affect each other by warping physical space. By notifying agents of their position changes, and causes, *Redux′* performs a coordination service, allowing the agents to adjust to each other's changes.

This is not to say that all of the design process is best represented as search. We do not address information search, conceptual analysis, or even the evaluation of options. We assume each person or computational agent will carry out these operations in an idiosyncratic manner. But we claim that the choice of an option and its relationship to other choices can be represented as search in a way that provides commonality and can be used to coordinate design among distributed agents. Further, this is the case even when not all of the options are generated at the time of choice, but perhaps dynamically as needed to resolve conflicts.

We illustrate this notion with different aspects of design search in this paper. However, we will not emphasize Redux terminology or search notions. First, that has been done elsewhere (Petrie, 1993). Second, it is more important that these search notions map onto engineering design process concepts than that they be explained in standard AI search terms. In Sec-

[1]This work was partially funded by Navy contract SHARE N00014-92-J-1833.

tion 4, we discuss how the Redux and First-Link models map. The significance of this is that the search notions do map. And, we show how the notions are used for important coordination services. The engineering design terms will be emphasized and are intended to be sufficient for understanding without more precise definition of the Redux terminology. However, we will begin with an overview of *Redux′* to avoid too much dependence upon previous papers.

2. *Redux′* Overview

The Redux formal model and its application to various domains is described in (Petrie, 1991) and summarized in (Petrie, 1992). This work has been used to reimplement the model as a configuration engine at the Universität Kaiserslautern in the domain of tool-setup planning (Paulokat and Ritzer, 1993).

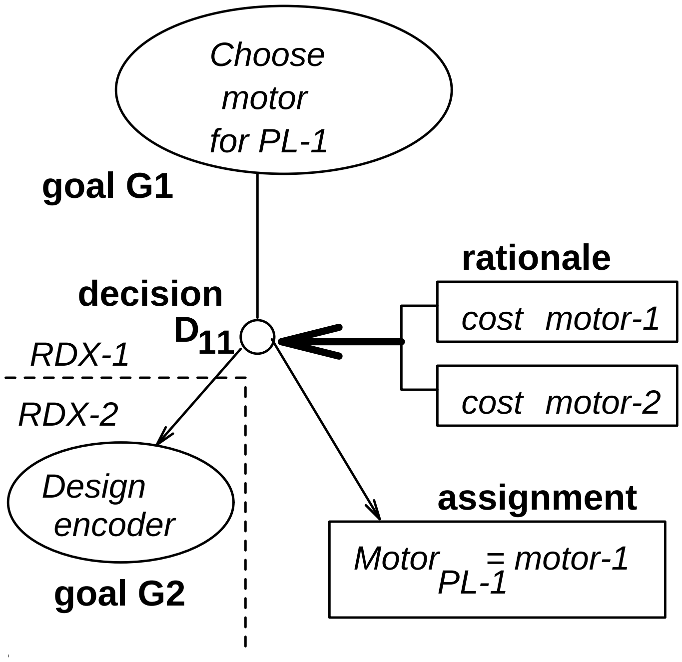

Figure 1. A Decision

Figure 1 illustrates the Redux notion of a design decision. In this example, taken from the PACT (Tenenbaum *et al.*, 1992) experiment, given a *goal* $G1$ of *Choose Motor* for planar manipulator *PL-1*, an engineer makes a choice of *motor-1*. The result of this decision, say D_{11}, is an *assignment* of *motor-1* to the “motor slot” of the design and perhaps a *subgoal*, say $G2$;

the design of the angle encoder and associated electronics required for such a motor.

While the notion of depth-first search where backtracking is caused by constraint violation is central to the Redux model, this example illustrates some important differences to other approaches. The first is that a *decision* may have a *contingency* associated with it. For example, in this case, the possible unavailability of the motor is an unexpected future event that would automatically invalidate the choice of motor.

The second is that decisions are also said to have a local *optimality*. This depends upon the validity of the decision *rationale*: why one operator was picked over the others in a conflict set. This reasoning is generally domain-specific[2]. Suppose there was one other possible choice, *motor-2*, and *motor-1* was chosen because it was cheaper. This *rationale* may or may not be recorded. If it were, it would consist of the costs for the two motors and is indicated by the bold arrow in Figure 1. If this rationale for the decision is invalidated, perhaps because of changing prices, Redux would propagate the loss of optimality, but would not automatically invalidate the decision.

The distinction between a contingency occurrence that automatically invalidates a decision and loss of optimality that only invalidates a decision rationale is important for management of the search space. Lack of this distinction has caused the failure of previous planners in the opinion of the authors. In this system, contingencies cause a plan to be immediately changed. Optimality loss is a notification that the decision should be only reconsidered, but that the design is still correct and consistent[3]. Automatic revision of a design due to a slight change in rationale, such as the cost of a part, might undo person-months of design.

Part of the decision rationale is a third difference between Redux and constraint-based approaches to design. In Redux, the rationale for backtracking is maintained and contributes to the optimality of a decision. In (Petrie, 1992), we explain the novelty of this approach and why it is equivalent to tracking the Pareto optimality of a design solution. While a full explanation is beyond the scope of this paper, the example in Section 5.3 illustrates the utility of this functionality.

Fourth, Redux differs from constraint-based approaches in that it treats subgoaling. One way of making progress in a design is to decompose the current problem into subproblems. In Redux, this is also a design decision, with a rationale, contingencies, and subject to backtracking. In fact, *Redux′*

[2] Redux does not treat the derivation of decisions, including trade-offs between alternatives: it only records the facts and rules used to make a decision.

[3] *Constraint violations* are a third source of change: these are noted along with a choice of decisions to retract. See Section 5.3.

is not a constraint manager and is indeed complementary to systems as (Bowen and Bahler, 1992) and (Darr, 1992).

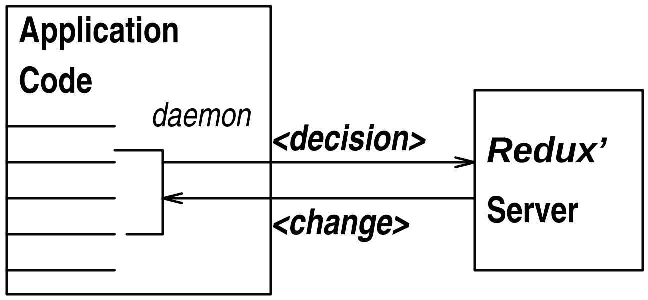

Figure 2. Decision Server

The *Redux'* model assumes that problem solvers are at least implicitly following this model of design decisions. These problem solvers can be viewed as design agents that send a *Redux'* server network messages about decisions and related types of objects. The decision in Figure 1 is defined by the *goal*, the *contingency*, the *rationale*, the *assignment*, and the *subgoal*. (A decision must have a goal and at least one assignment or subgoal.) These messages are sent either by the user or by daemons that "wrap" the application code. When selected lines of application code are executed, such daemons send the appropriate messages to the server. The output is other messages, representing the change propagation, sent to the user, or receptor daemons. This is shown schematically in Figure 2.

The *Redux'* model propagates the dependencies associated with the various objects as described in the formal model (Petrie, 1993). This results in notifications sent to either the applications or human designers that help to coordinate the design. In this paper, we give specific examples of these notifications rather than the complete and abstract model. We also attempt to describe the behavior of the system in engineering terms rather than in formal search terms as is usual in computer science.

3. Next-Link Overview

First-Link (Park *et al.*, 1994) is a distributed aircraft cable harness design system. In this application, we want to configure the electrical cable bundles that run throughout modern aircraft. Anyone that has ever seen the internals of an aircraft will be amazed that such a bewildering array of cables is designed, rather than organically grown.

Yet the cables are designed concurrently by different sets of engineers; e.g., there are package designers for schematics, cable designers for cable topology, model builders for routing and installation configuration, and structural designers for encapsulating structures and clamps. In addition, the same engineer may perform different tasks at different times: e.g., physical routing or selecting components such as shielding, plugs and sockets. Each of these major tasks is composed of various subtasks. All of them interact. Someone must also evaluate the final weight and cost of each configuration. And of course the design must be documented to accommodate changes throughout the life of the aircraft. The Lockheed AI Center is building a version of the First-Link system based on the research prototype developed at Stanford to assist engineers in generating and maintaining cable designs.

The First-Link domain has an appropriate level of complexity with which to test approaches to design coordination. It is not so complex as to take too long to show success, but it is sufficiently complex, with real problems and data, to demonstrate the advantages of distributed approaches, as well as test scaleability. And it is representative of a large class of configuration problems. First-Link is also advantageous because much work has been done already on the translation issues. The existing First-Link agents share a common domain ontology so that strings generated by one agent have meaning for the others. Finally, First-Link has been developed with an established agent protocol that serves as a benchmark for what functionality can be added by and at what cost.

Next-Link is a specific experiment that will test the *Redux'* approach to design coordination. We are modifying First-Link to reveal finer-grained tasks and their interactions and then do a better job of coordinating the agents and tasks. Analysis already performed shows that the cable design process stands to benefit dramatically from a simultaneous engineering approach and distributed functionality. Harness geometry, weight and cost are greatly affected by decisions made upstream in the superstructure design process. Redesign of the harness is necessitated by almost every change that affects the interior geometry of the superstructure. Such iterations are common because the development cycle is long and the design process must begin early to meet deadlines. Because there is no general support for effectively managing these iterations, cable harness design is usually expensive.

The general lesson for difficult configuration design is that backtracking is inevitable but thrashing is avoidable. Further, one wants not only to avoid old conflicts, but also to detect *opportunities* to return to old preferred choices because a conflict is no longer applicable. Such functionality applied to cable harness design has the potential to reduce iterations, reducing cycle time, improving the design, and saving money.

In the next sections, we will describe in detail how *Redux'* is being added to the First-Link model as part of the conversion to Next-Link. We will not detail the breakdown of the existing agents into finer-grained agents and their interactions, but rather explain here the computational nature of the interactions and the functionality the Next-Link will have due to increased coordination of the agents.

4. BASE Agent Interaction

Let us call the initial First-Link agent interaction model the "BASE" model. Each domain-specific agent sends "publish/request" types of messages to a Central Node(*CN*) that is a generic agent that facilitates and mediates communications. Each domain-specific agent has a set of capabilities according to the tasks it can perform. Each task has a set of inputs and outputs, both of which are called design *features*. Given all of the values of all of the input features for a task, an agent may then produce values for the task outputs.

In a "backward chaining" sequence of messages, one agent, say A_1, will want to perform task T_1^i (using the notation that T_j^i is Task i of Agent A_j)[4] and need all of the inputs for that task. A_1 will send *request-feature* messages to *CN* for each of these. Each agent has previously registered its capabilities with the *CN*. Thus the *CN* knows which of these inputs is an output of the task of some other agent, say task T_2^k of agent A_2. A value for this feature is then requested from A_2 by *CN*, unless A_2 has previously supplied the needed feature value. If *CN* already knows the value, it simply supplies it to A_1 with a *feature-data* message. But a *request-feature* message kicks off a similar process for task T_2^k by A_2

When A_2 has completed task T_2^k, it notifies *CN* of the availability of the output feature values with a *publish-features* message. For each output that matches some input of any other agent, such as A_1, the *CN* will send A_1 a *notify-new* message with the new set of features. Then A_1 will request the values of the features that match any of its task inputs for which values not yet known. Notice that if A_1 had not previously been working on some matching task T_1^i, the *notify-new* message will provoke it to do so. Thus the overall computation may proceed in a "forward chaining" fashion also.

The entire protocol sequence is illustrated in Figure 3. This protocol represents the current state of the First-Link system. In addition, when the *CN* is notified that a feature value has changed, any agent for which that feature is an input is notified to recompute the appropriate task; all previous outputs of that agent and task are considered invalid.

We would like to be able to map the Redux ontology onto existing design agents. The hypothesis is that the Redux model of design is sufficiently ubiquitous that the ontology can be retroactively "wrapped around" the

[4]In the current implementation, each agent only performs one task.

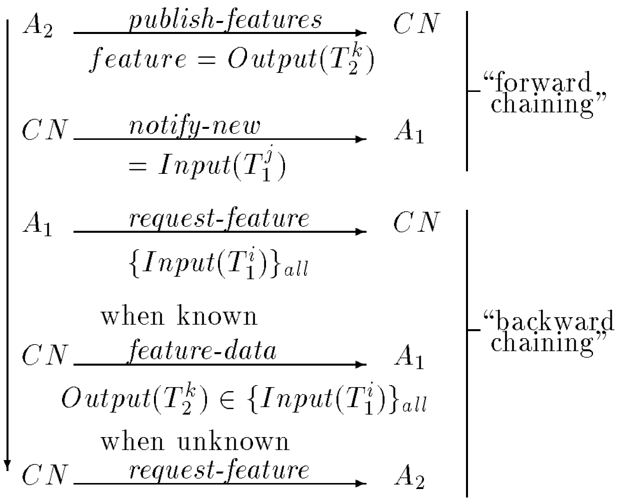

Figure 3. BASE Interaction Model

existing agents and messages. In fact, we can make the simple translation for terms and semantics shown below:

First-Link	Redux
Task T_1^j	Goal: to do T_1^j
Feature	Assignment Variable
Output value	Assignment
Input value	Dependent Assignment
publish-features	make decision
$Output(T_2^k)$	with assignment
notify-new	establish goal
$Input(T_1^j)$	to perform T_1^j
request-feature	dependent assignment
$A_1 \rightarrow CN$	between A_1 and A_2
$CN \rightarrow A_2$	for T_1^j, T_2^k

Thus, by allowing the *Redux'* agent to listen to the same messages that go through the CN, the former can convert these to the Redux ontology and semantics. Change in feature values is handled the same way by *Redux'* as by the CN, given this BASE model with no enrichment. The significance of this initial mapping is that *Redux'* can shadow the existing system, performing the same functions with the same messages.

5. Enriching the Model

There are several components of the design process the BASE model does not address explicitly, but which can be addressed by viewing the design process as the coordinated search of the design space. In this section, we describe the additional functionality of Next-Link in terms of agent interaction coordination due to *Redux'*. The first design process component we consider is subtasking.

5.1. SUBTASKING

If one task of an agent requires one or more subtasks to be accomplished by other agents before the supertask is done, a distributed design system should notify the owner of the supertask when all of the subtasks have been *completed*. An example subtask tree, taken from First-Link, is shown in Figure 4. In AI search terms, the satisfaction of subgoals needs to be propagated up the subgoal tree(s) in the search space. In concurrent engineering terms, the supertask isn't completed until the subtasks are.

This basic design search control function is not explicitly present in the BASE protocol. It is also not a part of the basic KQML (Finin *et al.*, 1992) protocol for distributed agents. This is an excellent task for a generic SHADE (Tenenbaum *et al.*, 1992) facilitator such as a *Redux'* agent. As shown by the small thicker arrows in Figure 4, *Redux'* propagates the completion of tasks up the subtask tree. This is especially important when the tasks are performed by different agents. In this example, agent A_1 needs to know when agents A_2 and A_3 have completed tasks T_2^1 and T_3^1, respectively.

Goal satisfaction, or task completion, is one dimension of the design search space that agents need to track. A second is whether or not a task is *in progress*. In search terms, this is whether or not a goal has been reduced. *Redux'* tracks this progress of a task. Notice that a designer need not be notified when he makes a decision to work on a task: that the task is then in progress is to be expected. The designer needs to know if and when that is no longer the case; there is unexpected *loss of progress*.

So *Redux'* tells an agent not only when a task is complete, but also when there is progress loss because of the decision of some other agent or some new fact. For example, the clamping configuration may no longer work if some other agent changes the cable geometry. This represents a *loss* of progress in the design. The designer must reattack the task of clamp configuration. Not only does A_3 need to be notified of this, but A_1 needs to know tasks T_1^2 and T_1^1 are no longer completed as previously thought. That is, there is also *completion loss*[5].

It may be that some design task is simply impossible to accomplish in the current state of design. Suppose, for example, the design cannot be proofed with the current clamping configuration. *Redux′* not only records such a *block*, but notes which decisions may need to be revised and which other tasks may be affected. In the example, the block of task T_2^1 by agent A_2 means that A_1 may have to redetermine design features used to perform task T_1^2, which may affect the way agent A_3 has performed task T_3^1.

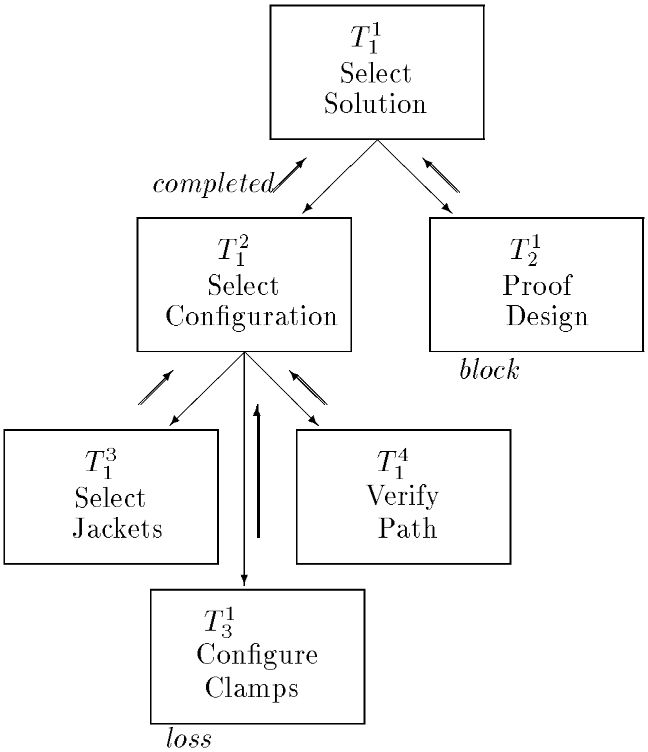

Figure 4. Subtask Tree

Finally, it can also happen that given subtask becomes *superfluous* because the supertask has been discarded. Whenever a subtask no longer has any valid justification, the owner of the subtask should be notified. The task of determining clamping for a cable branch will no longer be necessary if that branch has been eliminated by the cable topology manager. In search terms, goal validity, or invalidity, must be propagated down the subgoal tree(s). *Redux′* also tracks this third search dimension, notifying

[5]Since there is no completion loss without loss of progress, notifications of the latter are suppressed by notifications of the former.

the agents affected. This can also affect design feature assignments, since those generated for a superfluous task become redundant. For example, a particular clamp may no longer be required.

5.2. DECISION REVISION

In *Redux'*, tasks lose progress and completion, or become superfluous because of decision revision. A design decision is a decision to accomplish a task in a particular way (perhaps involving subtasks). The subtask tree in Figure 4 was developed by a series of decisions. An example is shown in Figure 5 in which task T_1^2 was not the only result of the decision. The particular configuration to be selected uses cable of single shielded wires, which is a design feature value, based on initial specifications.

If the decision is revised, the task is no longer in progress until a new decision is made. If a decision is revised, its subtasks become superfluous, and so do all the ones "below" them. Decisions may be revised during design for a variety of reasons. Perhaps a planned part is not in stock; e.g., the chosen clamps are not in the. This is a *contingency* that invalidates the decision to use the part. In search terms, this path has been rejected, as well as all of the paths below it. For concurrent engineering, one agent stating a fact such as the unavailability of a part may invalidate a second agent's decision. The latter must be notified, and the effects of the decision revision propagated to other agents.

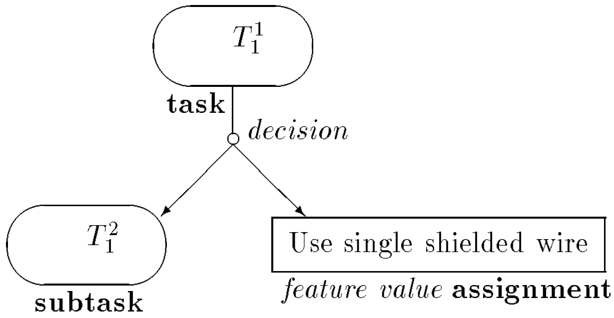

Figure 5. A Next-Link Decision

The rationale for the decision may also change. For instance, the agent in charge of the parts catalog updates the cost of the part, making it more expensive than previously believed by another agent that made a decision to use the part. Such change present designers with possible *opportunities* to improve the design. Perhaps old stock of Kapton insulation is actually cheaper now and should be used instead of the default insulation choice.

Or perhaps not. It is important that *Redux'* notify the designer agent of the need to reconsider the decision, but unlike the case of contingencies, not automatically retract the decision, possibly undoing much design work for little or no gain.

5.3. CONSTRAINTS

In this application, many other sorts of problems are best modeled as constraint violations. A structural change invalidates free space for the cables. A module or package changes size. A new heat or other electromagnetic radiation source is added for which current shielding is insufficient. Any of these changes may conflict with the current design, necessitating changes, such as cable segment rerouting or shielding changes. In the formal model, such conflicts are constraint violations and the remedies are resolutions of those violations. These are different from contingencies and loss of decision optimality described above. In this section we do not discuss how such constraint violations are detected, since there are a variety of well-known mechanisms, but rather the response to them and the effect on decision rationales.

In the Redux model, design decisions lead to exactly two kinds of results: subtasks and/or *assignment*[6] of design feature values, as shown in Figure 5. Design feature value assignments that have been decided have a status similar to subtasks. They may become "invalid" because the decision that generated them has been revised. In all cases, the designer will want to know the status of the design features, as well as the subtasks. *Redux'* will certainly track this status, but such change can also affect design rationale in an important special case.

Constraint violations occur when feature value assignments conflict. A constraint violation may involve as many agents as made the decisions that led to the design features that are in conflict. Each agent needs to be notified of the problem. A distributed system must identify the underlying decisions, and their makers, and then assist in the resolution of the conflict. The special case involves revision of a revision.

Suppose that two agents are involved in a conflict. It is decided that the first agent will revise his/her decision. Perhaps that agent chooses to use more expensive shielding for a cable that the second agent wants to route near a heat source. This is illustrated in Figure 6, where decision D_1^1 of agent A_1 conflicts with decision D_2^2 of A_2. Decision D_1^1 is retracted and decision D_2^1 is made instead. Now suppose A_2 later independently reroutes the wire away from the heat. The assigned placement of the cable next to

[6] Design feature "assignments" may represent a unique value, an interval or a region of configuration space, or some probability distribution about an expected value, depending on the domain-specific circumstances.

the thruster is invalid. The constraint no longer applies. *Redux′* will notify A_1 that the heat shielding is no longer needed.

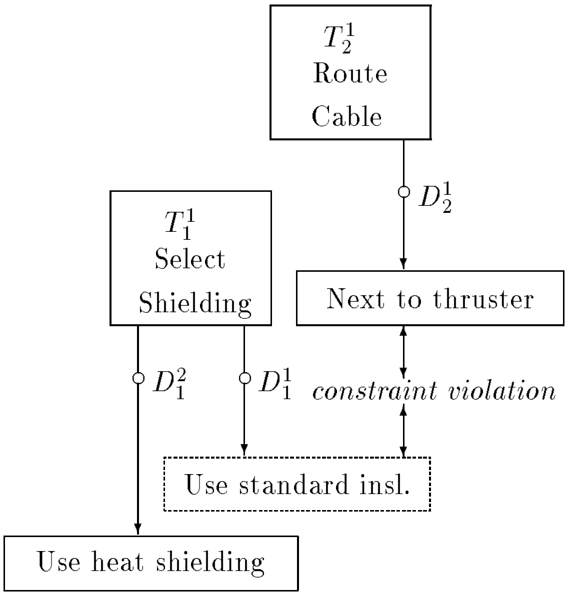

Figure 6. A Decision Revision

In this last example, *Redux′* is notifying the designer that he/she is going down one path in the design space and has forgotten the path originally chosen, which now should be reconsidered. *Redux′* makes the assumption that the designer prefers the first choice over the second and would like to know if ever it becomes again available.

6. Summary

The general case, with subtasking and design features, is that agents interact through a design space. The design decisions of one agent may change the global positioning of another agent within this space. By notifying each agent of unexpected changes, and answering questions about location in this space, the distributed system can help the agents coordinate their actions.

Redux′ assumes that tasks are initially neither in progress nor completed. In fact, it assumes no action has yet been taken, so no constraint violations or blocks exist. It will accept no decision for which a contingency already exists or with an invalid rationale. Thus, the sort of changes that will cause notifications are *task completion, completion* or *progress loss, superfluous tasks, constraint violations, blocks*, and *invalid decision rationales.* These are all unexpected changes in the design space.

It is important to note that *Redux′* functions as a central node with the other Next-Link agents but with no understanding of the cable harness design domain. It can nevertheless coordinate domain-specific agents, because it works with the Redux model of the design process to which each Next-Link agent conforms. This conformation was not designed into the agents. It results simply because the search-based model is generic to at least the configuration design domain. Thus *Redux′* messages can be "wrapped" around the existing design decisions with minimal modification of the existing code.

Finally, in distributed design, backtracking is inevitable but thrashing is avoidable. *Redux′* not only eliminates design iterations due solely to thrashing,[7]. but ensures that known opportunities for improving the current design are not overlooked during iteration, while allowing tasks and subtasks to be pursued in parallel.

References

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

Darr T., Birmingham W., "Concurrent Engineering: An Automated Design Space Approach," Tech. Report CSE-TR-149-92, U. of Michigan, Ann Arbor, MI.

Finin, T., McKay, D., and Fritzson, R., "An Overview of KQML: A Knowledge Query and Manipulation Language," Technical Report, Computer Science Dept., U. of Maryland, 1992.

Gebala, David and Steven Eppinger, "Methods for Analyzing Design Procedures," DE-Vol. 31, *Design Theory and Methodology*, ASME, 1991.

McGuire J. et al., "SHADE: A Medium for SHaring Design Knowledge among Engineering Tools", *Journal of Concurrent Engineering: Applications and Research* (CERA), to be published.

Park, H., M. R. Cutkosky, A.B. Conru and S-H Lee, "An Agent-Based Approach to Concurrent Cable Harness Design," *AI EDAM*, **8**, No. 3, 1994 (in press).

Paulokat, J. and H. Ritzer, "Unterstuetzung der Kontrolle bei der Konfigurierung durch ein problemklassenspezifisches TMS," *Proc. 2. Deutsche Tagung Expertensysteme (XPS-93)*, F. Puppe, A. Guenter (Eds), Springer, 1993.

Petrie, C., "Planning and Replanning with Reason Maintenance," Dissertation. MCC TR EID-385-91, December, 1991.

Petrie, C., "Constrained Decision Revision," *Proc. 10th Nat. Conf. on AI*,San Jose pp. 393-400, AAAI Press, July, 1992.

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

Tenenbaum, J., J. Weber, and T. Gruber, "Enterprise Integration: Lessons from SHADE and PACT," *Enterprise Integration Modeling.*

Toye, G. et al., "SHARE: A Methodology and Environment for Collaborative Product Development", *Proc. 2nd WOrkshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Press, pp. 33-47, 1993.

[7] We also plan to study reductions of design iterations along the lines of (Gebala and Eppinger, 1991)