From: AAAI Technical Report WS-93-07. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

Design space navigation as a collaborative aid

C. Petrie
MCC Enterprise Integration Division
3500 West Balcones Center Drive
Austin TX 78759
USA

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 *Redux'* services and the mapping to existing agents are illustrated by experience with the *FirstLink* distributed design system.

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

1 Introduction

The *FirstLink* project in the Stanford Center for Design Research is sponsored by Lockheed in order to develop a distributed system to design cable harnesses for aircraft[2]. The design domain is typical of the generic configuration task. but also tackles tough geometric modeling problems. The initial prototype of the distributed design system has a simple agent interaction model that we are augmenting with *Redux'*[4], an agent implementing a subset of the full Redux model described in [3].

The Redux model is based upon AI problem solving notions of search spaces. 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.

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[4]. Second, it is important that these search notions map onto engineering design process concepts. In Section 2, we discuss how the Redux and FirstLink 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. But the engineering design terms will be emphasized and are intended to be sufficient for understanding without more precise definition of the Redux terminology.

2 BASE Agent Interaction

Let us call the initial FirstLink agent interaction model the "BASE" model. Each domain-specific agent sends "publish/request" types of messages to a Central N-ode(*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 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.[2]

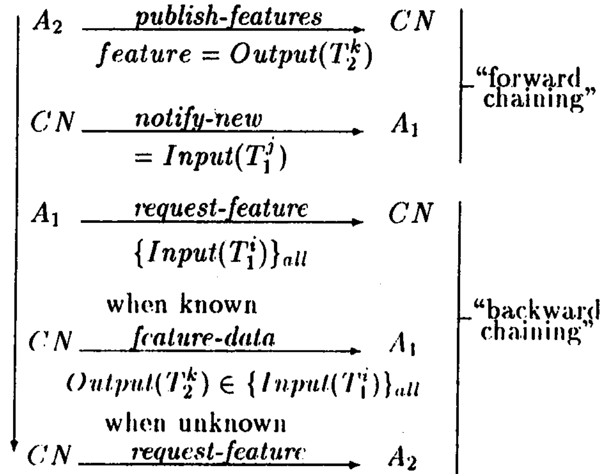

Figure 1: BASE Interaction Model

The entire protocol sequence is illustrated in Figure 1. This protocol represents the current state of the FirstLink 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 existing agents and messages. In fact, we can make the simple translation for terms and semantics

[2] This overall design is due to Andrew Conru.

shown below:

FirstLink	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.

3 Enriching the Model

However, the BASE model of FirstLink interactions is not sufficient for collaborative design. There are several components of the design process this model does not address. Here we show that some of these are addressed by viewing the design process as the coordinated search of the design space. The first design process component we consider is subtasking.

3.1 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 FirstLink, is shown in Figure 2. 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 present in the BASE protocol. It is also not a part of the basic KQML[1] protocol for distributed agents. This

is an excellent task for a generic SHADE[5] facilitator such as a *Redux'* agent. As shown by the small double-lined arrows in Figure 2, *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*[3].

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.

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 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.

[3] In fact, since there is no completion loss without loss of progress, notifications of the latter are surpressed by notifications of the former.

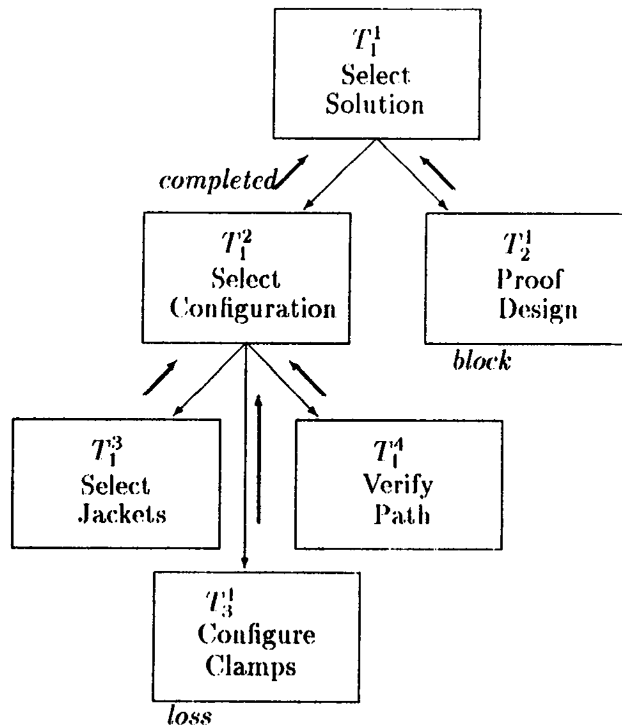

Figure 2: Subtask Tree

3.2 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 2 was developed by a series of decisions. An example is shown in Figure 3 in which task T_1^2 was not the only result of the decision. The particular configuration to be selected uses solid copper cable, which is a design feature value, based on initial specifications.

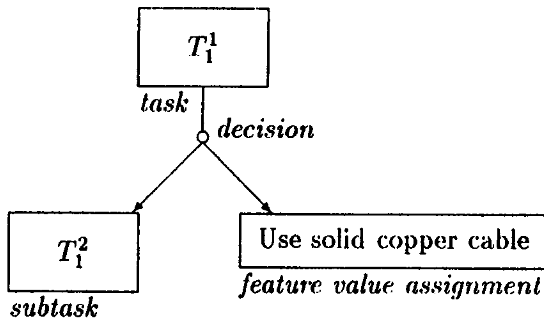

Figure 3: A Decision

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., there is not enough copper cable. 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.

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 twisted copper is actually cheaper now and should be used.

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. In any case, this is another type of change in the shared search space that may need to be propagated among agents.

3.3 Constraints

In the Redux model, design decisions lead to exactly two kinds of results: subtasks and/or *assignment* of design feature values, as shown in Figure 3. 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 4, 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 the engine is invalid. The constraint no

longer applies. *Redux'* will notify A_1 that the heat shielding is no longer needed.

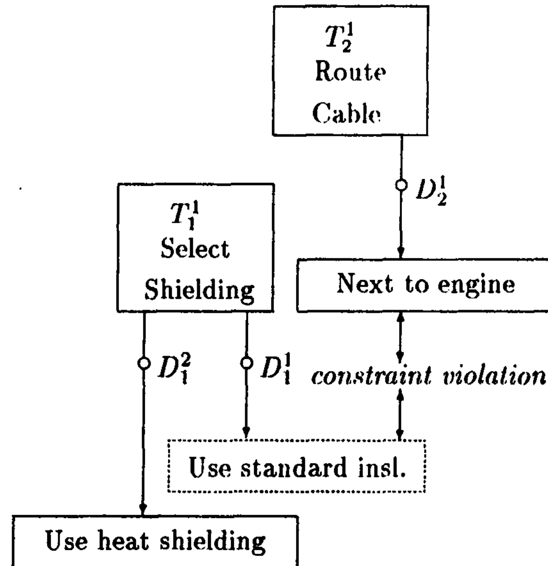

Figure 4: A Decision Revision

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.

4 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 contigency 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.

Finally, we note that *Redux'* functions as a central node with the other FirstLink 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 FirstLink 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.

References

[1] Finin, T., McKay, D., and Fritzson, R., "An Overview of KQM-L: A Knowledge Query and Manipulation Language," Technical Report, Computer Science Dept., U. of Maryland, 1992.

[2] Park, H., M. R. Cutkosky, A.B. Conru and S-H Lee, "An Agent-Based Approach to Concurrent Cable Harness Design," Tech Report CDR 19930217, Center for Design Research, Stanford University, Stanford, CA, 94305.

[3] Petrie, C., "Constrained Decision Revision," *Proc. AAAI-92.* Also MCC TR EID-414-91, December, 1991.

[4] Petrie, C., "The Redux' Server," *Proc. Internat. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, May, 1993. Also MCC TR EID-001-93, January, 1993.

[5] Tenenbaum, J., J. Weber, and T. Gruber, "Enterprise Integration: Lessons from SHADE and PACT," *Enterprise Integration Modeling*, C. Petrie, ed., MIT Press, October, 1992.