Agent-Based Process Management

Charles Petrie
Center for Design Research,
Stanford University
petrie@stanford.edu

Sigrid Goldmann
Andreas Raquet
AG Künstliche Intelligenz,
Universität Kaiserslautern
Kaiserslautern, Germany
sigig, raquet@informatik.uni-kl.de

Abstract

Integrated project management means that design and construction planning are interleaved with plan execution, allowing both the design and plan to be changed as necessary. This requires that the right effects of change need to be propagated through the plan and design. When this is distributed among designers and planners, no one may have all of the information to perform such propagation and it is important to identify what effects should be propagated to whom, and when. We describe a set of dependencies among plan and design elements that allow such notification by a set of message-passing software agents. The result is to provide a novel level of computer support for complex projects.[1]

A longer version appeared in the *Proc. of the Intnl. Workshop on Intelligent Agnets in CSCW*, , Deutsche Telkom, Dortmound, Sept. 1998, p. 1-17.

1 Introduction

Today, traditional project management methods are not sufficient for managing the many tasks in the design and development process. They do not take into account all the sources of change, the task interactions, and the necessity for distributed planning. They do not provide proper **change notification**: notifying the right agents (people or software) of the effects at the right time in the process.

Process coordination is always most complex in domains that require artifact design and construction planning. This is particularly true when the artifacts are large and many people and software tools must be coordinated and managed.

When design changes cause plan and schedule changes, the problem is worse than simply modifying the design. Somehow, all the people assigned to the affected tasks, and no one else, should be notified of the change and how it affects them. This difficulty is reflected in the expense of coordinating projects and in the achievement of suboptimal results[Benda, 1998; McKinsey, 1997].

1.1 Example Problems

The following few examples serve both to illustrate this difficulty and to suggest the kind of questions answered by this paper.

"Fast track" construction attempts allow the architect to finish or change the design after construction has started. These design changes frequently necessitate a change in the construction plan and or schedule. As a simple example, suppose that the design calls for concrete roof tiles. Then the wall plaster must be applied after the roof is built, or the heavy tiles will deform the walls causing the plaster to crack. But if it is decided later that lighter fiberglass tiles should be used instead, it is no longer necessary to wait to plaster the walls - the two tasks can be more concurrent and the plan schedule shortened. What kind of computer support could ensure that when the architect changes the material specification, the contractor will be notified of the possibility of shortening the schedule?

Suppose that we are designing and building a prototype of a new gyroscope for use in navigation equipment. Suppose further that one engineer decides that high resolution encoders is a better design than the rate sensors plus low resolution encoders in the current design. How can we ensure that the machinist who is designing the frame to hold the components will be notified of this change?

A hospital adds a new surgery wing. The architectural specifications for the width of the doors into the new wing were determined by the width of the hospital beds that had to be wheeled through them. During construction of the new surgery, someone in the hospital decided independently to buy new, much wider, beds. What kind of process mechanism could have avoided the subsequent remodeling of doors in the freshly built surgery?

Imagine building the international space station with hundreds or even thousands of companies and engineers and contractors of all sorts. How can the emerging mass

[1] This paper is a short version of "Agent-Based Project Management",
Lecture Notes in AI 1500, Springer Verlag 1999, available at http://cdr.stanford.edu/ProcessLink/papers/DPM/dpm.html

of design decisions and changes be coordinated across organizations? How could this ever be "fast-tracked"?

The problem is not lack of connectivity. With the Internet, and intranets, combinations of email and groupware enabling everyone to reach everyone else with ease, one could make a case that the ability to task each other so easily is actually making the problem worse. The problem is that there insufficient structure supporting the distributed task interactions of modern enterprises, especially for project management.

1.2 What is Missing

Today's distributed project management tool are still based upon a single-user model of planning and the change notification is still primitive- meaning that all change notifications must be pre-specified, usually by the users.

If one considers single-user project management tools, such as *MacProject* or *MS Project*, one can see immediately that they are completely inadequate. The model is that a single general contractor makes decisions and changes and then somehow notifies the people involved. Notifications to the general contractor that cause changes and the notification of the people affected by the change are simply not part of the computer support, though very much a part of any project. The only distributed support from such products is that emails may be sent when a task is assigned and completed.

AutoPlan{5} is in contrast a distributed project management tool that provides an electronic blackboard for change notification. This allows people to receive email when a pre-specified type of event occurs, such as assignment of a task. This definitely takes the computer support of Distributed Project Management (DPM) one step further, but it is still based upon a single-user model of planning and the change notification is still primitive. That is, the general contractor is still responsible for all changes and all of the change notification must be pre-specified, usually by the users.

In small or medium sized projects, a general contractor may track the hundreds of informal change orders with a cork bulletin board and notes. That we do not provide better support for such projects limits the complexity of the project to that which can be managed by a single person. As a result, plans are kept simple and rigid and many opportunities to improve the plan, or even to avoid mistakes, are missed.

But worse is the cumbersome formal change order process required on large engineering projects, which multiple levels of management approvals, with increased time and cost, in an attempt to catch most interactions. Usually, many more people are notified for a given design change than are actually necessary, burdening the whole design process.

The current lack of technology for coordinating design decisions and managing change over the life of a product creates higher costs, longer cycle times, and poorer quality than is currently possible. Occasionally, this lack of technology is even dangerous, whether one is maintaining an older passenger plane or decommissioning a nuclear weapon.

1.3 Distributed Integrated Project Management

As described in more detail in the white paper "Distributed Integrated Process Coordination"{9}, process coordination is missing from various well-known computer support approaches to business integration, group collaboration, and project management. Process coordination means the runtime determination of who should be notified and what task should be performed next. Computer support for this means some automation of the notification and tracking of task properties as they are created or change.

Process coordination is more general than workflow in that it does not require that all tasks and ways of doing them be identified prior to process execution. For instance, anyone should be able to assign any tasks to anyone else at any time during process execution. And appropriate notifications associated with that task delegation, or subtask assignment, should be handled by the computer support system.

Process coordination is most complex in domains that require artifact design and construction planning, especially when the artifacts are large and many people and software tools must be coordinated and managed. *Distributed Integrated Project Management* **(DPIM)** is an extreme form of Process Coordination in which design, planning, scheduling, and execution are interleaved across distributed organizations and engineering disciplines as well as computer tools.

In particular, we want to be able to support *change that occurs from incomplete designs*, in order to support "fast tracking", as well as *contingencies and planning under uncertainty*. We also want to be able to support *design and planning that is distributed among people and software* that can best solve parts of the problem.

Computer support for DPIM necessarily involves a heterogeneous mix of software and people passing messages describing tasks and changes. The least commitment strategy, with respect to platform, of federating people and their software tools is to use an *agent communications language* (ACL) such as KQML{6} or FIPA{7} . This only supposes that each software module, possibly just an interface for a person, can exchange ASCII text messages according to standard Internet protocols such as socket connections with TCP/IP.

If the software systems are sufficiently homogeneous so that they can make a stronger commitment, they can directly exchange objects using a facilities such as Java RMI and CORBA. An standard ACL. using an agent[Petrie, 1996] model, requires less of a commitment to transport mechanism and more of a commitment to message semantics and protocol. In any case, the choice of the interoperability infrastructure is necessary but not sufficient for coordination and management of distributed projects.

What is also necessary is that the human and software agents also agree upon a *model of coordination.* This is also consistent with the view of agents as software and humans that share a common protocol of messages in which some responses are legal and some are not[Haddadi, 1996].

Because the central problem of distributed interleaved planning is *change propagation*, we characterize our coordination model as a *logical set of dependencies* among the project elements that can be used to determine the effects of changes within the project. This paper defines a such set of dependencies that should be managed by a computer system in order to coordinate a distributed project.

2 Dependency Key Ideas

This paper uses "design" to mean generally the design of the target artifact. A "plan" is generally the determination of the tasks and subtasks required to build the artifact, including the durations and other task features, except for the start and end dates, which are assigned by the "schedule". "Execution" will generally refer to the execution of the plan, which will change the state of the developing artifact.

Thus, the architect designs a building, for which a general contractor develops an overall plan and schedule for constructing. As concrete is poured, the plan is executed and the building takes shape.

We note here briefly, and explain further later, that these simple notions can be endlessly complicated. For instance, design itself can be planned, and the execution of such plans results in a design. Then we plan to build the design (which is sometimes called, confusingly, the "execution" of the design.) Anyone who decides how much time should be alloted to developing a part of a design has done design planning to some extent. However, we refer to the simpler uses, defined at the beginning of this section, of design, planning, scheduling, and execution in the discussion of dependency requirements below.

2.1 Precedence is Not Enough

Projects follow plans about tasks, their durations, and resources. These factors always implicitly depend upon the design of the artifact being constructed. As the design changes, these factors may change, because of *interactions between the design and the plan and schedule.*

One of the most important of these factors are the precedence relationships. These are usually functions of resources and other interactions of tasks. Yet such interactions are rarely captured. Traditional project management requires precedences to be input at the beginning of the plan. Therefore there can be no dynamic adjustment of the plan as factors that influence precedence change.

The roof tile and plastering example mentioned previously illustrates this. The two tasks of roofing and plastering are connected by the design feature of the weight of the roofing material. The precedence that the plastering task should follow the roofing task depends upon this weight. If this weight changes because of the design change, then the precedence ordering should be reconsidered.

This is an example of how a change in the design of the artifact may introduce an opportunity to improve the plan for constructing it. But this opportunity would be lost if precedence relationships are static and not tied to features of the design.

A key insight here is that the way to relate the artifact design and the plan is by including *artifact features and conditions in the plan.*

2.2 Plans are Designs Too

This suggests the next insight: that plans and schedules are also designs. Artifact designs can be characterized by design decisions about components and features. Plans and schedules are also designed artifacts with components and features.

A fundamental component of a plan is a *task.* Fundamental task features that need to be taken into account are *task inputs and outputs.* If one task outputs something that is an input for a second task, then at least there is a precedence relationship that the second task cannot end before the first. Thus, the *plan* design decision, which we will call a *planning decision*, was to have the second task follow the first, with a decision decision based upon a design rationale consisting of the task inputs and outputs.

The idea of design decisions gives us a way to think about *interactions between task planning, scheduling, and task execution.* If we can determine a general model for design change propagation, then we can apply it to planning and scheduling and interleave all three as is required.

And if this model allows concurrent design, then since planning and scheduling can be viewed as design, then the model should support distributed planning and scheduling as well.

3 The Redux Design Model

There is a general model of design change propagation: Redux[Petrie, 1993]. Our approach is to extend it to planning and scheduling, identifying specialized extensions, and allowing for special conditions that may arise as a plan is executed.

The first cut of these extensions was the Procura design[Goldmann, 1996]. This has been furthered by the forthcoming diplom thesis of Raquet and discussions with the developers of ComMoKit[Dellen, 1997].

3.1 Redux Overview

We begin this description with some general observations about Redux dependencies.

The basic Redux model is similar to that of gIbis[Conklin, 1988]. However, whereas gIbis is a passive recording of issues, Redux computes the propagation of changes to the design. By adding simple notions of conflicts and rationales, very useful inferences can be derived as shown in [Petrie *et al.*, 1995].

In addition to simply documenting design rationale, Redux makes the design rationale active by tracking its validity in several respects and notifying designers when it changes.

We have found that even in small design projects, people lose their ability to maintain a comprehensive picture of the history and interplay of design decisions, constraints and rationales. Under these conditions the advice generated by Redux' soon becomes non-obvious and intuitive only in hindsight as Redux' reveals forgotten opportunities and hidden conflicts.

Design Decisions

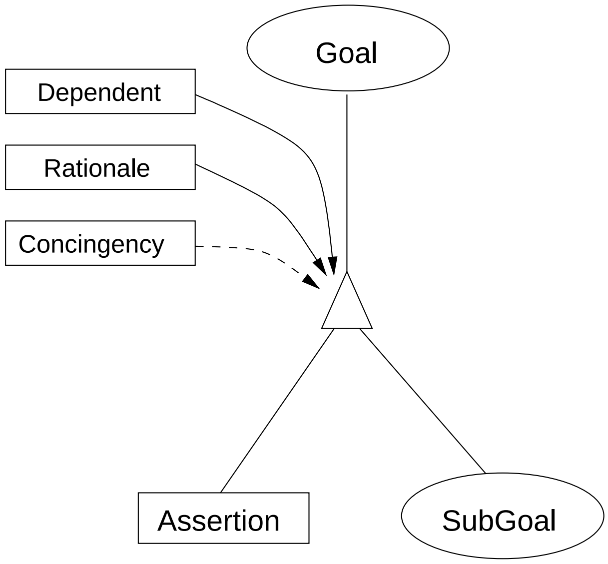

Figure 1: Redux structures

The Redux model of design is a model based on a simple notion of design decisions that could be used in a distributed design project such as PACT[Cutkosky *et al.*, 1993]. First a *decision* must be about some *goal*: this can be any sort of design task, issue, or question. The purpose of the decision is to somehow address this goal. For example, a design goal might be "design a computer".

Further, a decision must include some result of one or both of two kinds: an *assignment* and/or a new goal that is a *subgoal* of the original goal. The assignment is some statement about the design such as "the computer will use a fast chip". The assignment may optionally be structured so that it has a *feature*, such as "part-of"; an object, such as "memory-1", and a value, such as "fast memory chip mcp-2", resulting in a complete assignment such as "part-of memory-1 fast memory chip mcp-2". A subgoal is some new design goal that is necessary to satisfy before the original goal can be satisfied, such as "design the memory controller".

This model does not assume the existence of a single root goal and allows design to be represented in general as an arbitrary set of goals to be achieved. But in general, a goal is satisfied when all of its subgoals are satisfied so it is easy and useful to represent a design problem as beginning with a single top-level root goal, such as "design the computer".

If *reasons* are provided for a decision, Redux uses them to generate a decision rationale. If one of these reasons subsequently becomes invalid, then the decision rationale may become invalid, if there are no other supporting valid reasons. If the rationale becomes invalid, the agent that made the decision is notified that the decision should be reconsidered. The decision is *not* automatically retracted in this case.

For example, a decision rationale might include the two reasons "this memory costs 2 cents" and "this memory supplier is reliable". If these are two separate reasons, and the cost changes, then the decision to use this memory still has a valid rationale consisting of the supplier reliability. If this second condition changes, the decision maker would be notified that the decision should be reconsidered. If desired, both facts could be included as a conjunction comprising a single reason and the decision maker would then be notified if either changes.

Dependents and *contingencies* may additionally be associated with a Redux decision. A dependent is an assignment that is the result of some other decision. It can also be called a "decision input". A contingency is some environmental condition that is assumed not to be the case. If the dependent becomes invalid, or the contingency becomes true, then the decision is automatically retracted and becomes invalid. An example of a contingency is a statement such as "this memory part is unavailable".

Figure 1 shows a goal, that is decomposed to a subgoal and a assignment. Goals are represented by ellipses, decisions by triangles and assignments and facts by rectangles. justifying dependencies are represented by a solid line, retracting dependencies by a dashed line.

Multiple design decisions conflict with each other via their assignments. A set of conflicting design assignments is expressed as a *constraint violation*, though Redux says nothing about how such violations are detected and does no constraint propagation. However, given a constraint violation, Redux will determine which among a set of possible decisions might be rejected in order to resolve the violation. If a decision is rejected, it is *retracted* and a reason for the *rejection* is noted.

If a decision is retracted, the decision become invalid. Redux can also maintain reasons for the retraction. A decision can be arbitrarily retracted, but in general it will be rejected, meaning that there are good reasons for having rejected the decision, such as "this kind of memory doesn't work with this cpu".

A decision may become invalid if retracted/rejected or if an dependent becomes invalid or if a contingency occurs. in this case, all of its assignments and subgoals may become invalid, if they are not supported additionally by some other decision.

3.2 Redux Notifications

The simple Redux model allows inferences to be drawn that are useful in the notification of participants. There is an implemented Redux agent in ProcessLink that does just this. The essential requirement is that all agents notify the Redux agent when decisions have been made, describing the decision as above. The agents should notify Redux of constraint violations, goal blocks, and general decision rejections as well. Finally, facts may be asserted and deleted as desired. Changes will be reported according to the Redux model.

If a decision becomes invalid, then its objective goal is no longer reduced. The *decision maker should be notified at least that progress has been lost in working on this goal.* If the goal was previously satisfied, then the decision maker will be notified that *the goal is no longer satisfied.* If this goal satisfaction previously contributed to the satisfaction of some supergoal, then the decision maker for the supergoal will be notified of the loss of satisfaction of that goal.

If a decision becomes invalid, some of its assertions and subgoals may become invalid. The invalidity of these assignments and goals will have further ramifications. If a goal becomes invalid, then any decision maker for that goal should be informed that *the goal is now redundant* and any decision reducing that goal is now suboptimal as a result.

If an invalid assignment was used as dependent to a decision, then that decision becomes invalid, with the same notifications as above. If the assignment was used as a rationale, then *the decision becomes suboptimal* and the decision maker is informed the decision should be reconsidered.

If some decision was previously rejected in part because of of an assignment that is now invalid, the decision maker will be informed that *this decision may now be optimal* and should be reconsidered. The same is true for any fact or constraint in the rejection reason that becomes invalid.[2]

If some decision maker has attempted to make decisions about a goal and each has been rejected or invalidated, and the decision maker cannot think of a new way to work on the goal, the decision maker may declare a *goal block*. All of the decision makers responsible for an assignment in any of the rejection reasons or dependents for the previously defeated decisions will be notified of the impasse, as will the decision maker that created the blocked goal, if any.

3.3 Distributed Design Example

In order to show how the Redux model works with distributed design, we present a simplified version of work done recently with Toshiba.

Figure 2 represents a Redux agent working with four design agents in a project to design an optical device.

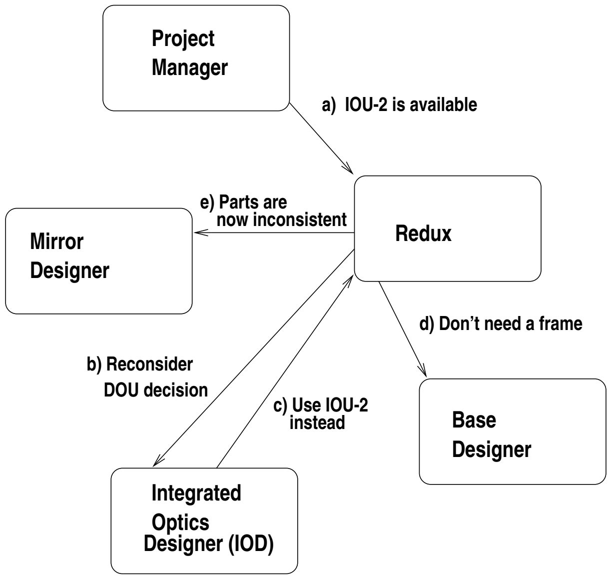

Figure 2: Design Agents

There is an overall Project Manager that sets the top-level goals and resolves disputes, a Mirror Designer that designs the mirror systems, the Integrated Optical Designer (IOD) that chooses the optical detector mechanisms, and a Base Designer who designs the base that holds the whole array of components.

During the course of the design, the IOD has tried to use two varieties of an "integrated optical unit" (IOU) but ran into problems in both cases, in one case simply because inventory said the part was not available. The IOD ended up using a "discrete optical unit" (DOU) which necessitated the Base Designer building an extra frame to hold the discrete components.

Using Redux, the Product Manager can examine the rationale for the DOU decision and see that in one case, a particular part, an IOU-2, was not used because inventory said it was unavailable. However, the Product Manager knows that this was an arbitrary status based upon project priorities, and declares the part available in message **a)** to Redux.

Redux knows to notify the IOD to reconsider using IOU-2 instead of the DOU in message **b)** in the figure. The IOD makes this change in message **c)** and in messages **e)** and **d)**, Redux notifies the Mirror Designer that the mirror parts decision uses the DOU information that is no longer valid, and the Base Designer that the goal of designing a frame for the discrete DOU parts is also no longer valid.

Such a system is implemented and running using the JAT*Lite* agent infrastructure{10} and Java applets that display colored decision graphs such as in Figure 3 using the KQML messages and extensions documented at {13}. This system has the great advantage that the applets can be downloaded to any Java-compatible browser anywhere on the Internet and the outstanding messages for that agent read and processed.

---

[2]This implements a version of Pareto optimality[Feldman, 1980] tracking described in [Petrie *et al.*, 1995].

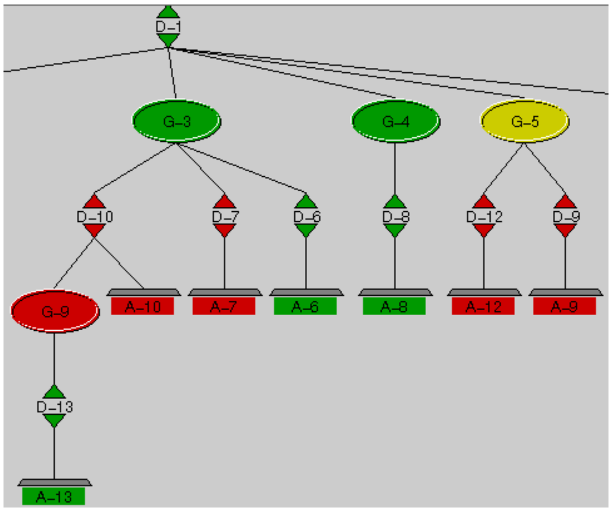

Figure 3: Agent Applet Graph

In Figure 3, the decision to use a DOU is *D-10* and the decision to use IOU-2 is *D-6* and the goal to build the discrete component frame is *G-9*. Since the part unavailability was a contingency that prevented the use of that part in attempted decision *D-6*, and since Redux automatically constructs a decision rationale for further decisions *D-7* and *D-10* based upon *D-6*, Redux can send message **b)** to the owner of *D-6*, the IOD. The goal *G-9* was a subgoal of *D-10* and so was automatically invalidated when that decision was rejected. This caused *D-13* to be suboptimal and message **d)** was sent to that decision owner, the Base Designer. Further, an assignment of *D-10* was used as a dependent by a decision of the Mirror Designer in choosing parts, so Redux invalidates that decision and sends message **e)** that the mirror parts goal is no longer satisfied.

This particular scenario is detailed on the web at {11}. Engineers enter these decisions either directly using a desktop applet agent, or indirectly by using CAD tools that have been "wrapped" to become ProcessLink agents{10} or by using design documentation authoring tools that allow Redux annotation{12}. A detailed examination of Redux rationale maintenance in another application is available in [Petrie *et al.*, 1995], in which the Redux support for resolving conflicts and maintaining a design rationale using dependency-directed backtracking is emphasized (and not otherwise discussed in this paper).

Now we would like to extend this design change notification functionality to project management.

4 Planning Dependency Extensions

In order to extend this Redux model to planning and scheduling, we will posit other agents specialized for planning and scheduling that will make special requests of Redux and require some extensions to the general model.

In the ProcessLink system, we provide a framework of domain independent agents of which one is Redux, one is a Constraint Manager (CM) that manages constraint solvers and performs constraint propagaton, and on is a Plan Manager (PM) that performs global tracking of the plan elements, using the Redux and the CM, and the JAT*Lite* agent infrastructure as a general "bus" for the exchange of messages. Domain-specific design, planning, and scheduling agents may connect as desired from anywhere on the Internet. These agents can also make Redux decisions and work with the PM and the CM using the ProcessLink Electronic Project Language (EPL) protocol{13}.

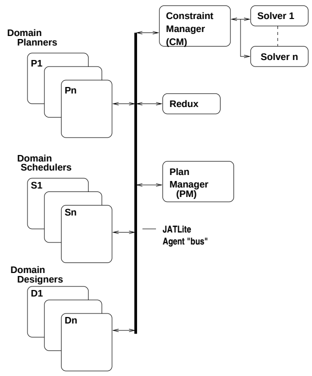

Figure 4: The ProcessLink Framework

One simple extension is that Redux especially treats assignments that have a feature with the keyword *"assign-agent"*. Then the feature object is expected to be the name of a goal and the value is expected to be the name of an agent. Given the new validity of such an assignment, when the goal is valid, the agent is sent a message that *the goal has been assigned to that agent.* When either the goal or the assignment become invalid, the agent is advised of this change.

However, most planning and scheduling extensions will be handled by the PM either by using the decision model to establish the right dependencies to be maintained by Redux, and the CM, or maintaining the dependencies itself. In addition to the dependencies described below, the PM must provide common project management functions, including resource management and representation and scheduling. We do not address these problems here as they are well understood and may be handled by a number of algorithms by the PM or other planning and scheduling agents.

4.1 Interleaving Designs and Plans

The first way in which Redux dependencies can be used effectively is to ensure that the plan does not contain valid elements that are unnecessary. We can do this by using plan subgoal validity.

In Figure 5, the design decision to use rate sensors and low-resolution encoders directly generates a plan goal to acquire the encoders and sensors, and machine the base

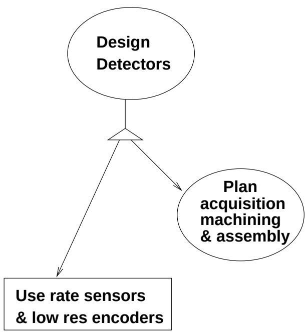

Figure 5: Plan Subgoal of Design Goal

for them and assemble them. This goal will become invalid as soon as the design decision is rejected in favor of high resolution encoders. This behavior is what is desired. Any planning decisions made based on this plan goal will become suboptimal and the planner notified, though the decisions will not automatically be invalidated. There will be possibly many subgoals and decisions beneath this high-level plan goal that should be carefully handled given a change.

However, this simple model is flawed because the super goal of the plan goal is a design goal, not another plan goal. Apart from the strangeness of the naming conventions, this has two disadvantages. First, it necessarily puts the designer in the position of generating plan goals, and two, it is now not possible to use the Redux goal satisfaction mechanism to determine if either the design or the plan alone is complete, without a lot of awkwardness.

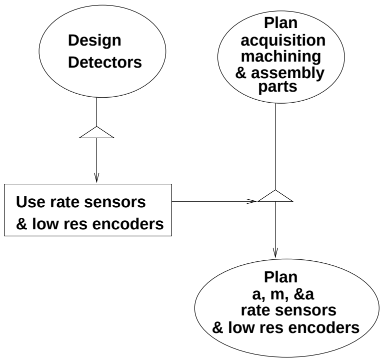

Figure 6: Design Rationale for Plan

What is going on here is that we have conflated the goal dependency with the goal generation. We need not generate the plan goal directly as a subgoal. We can instead posit the existence of a planning agent that reviews the design and decides independently on when and how to plan the execution of the design. It is only necessary *that the planning goals should become invalid when the planned parts are no longer a valid part of the design.* The easiest way to do this is to make design assignments a part of either the rationale or the dependents of planning decisions as in Figure 6.

The most automatic alternative is to make the design assignment a dependent so if the design decision becomes invalid, and the assignment becomes invalid (no other valid design decision supports it), then the planning decision to plan for those components will become invalid.

However, notice that this scheme a) biases in favor of the designer, and b) automatically invalidates any plan subgoals. Another choice for representation is to use the design assignment as only a rationale. This also results in a notice to the planner, but does not invalidate any subgoals or change the plan in any way. The planner and the designer can then argue about whether the plan or the design takes precedence.

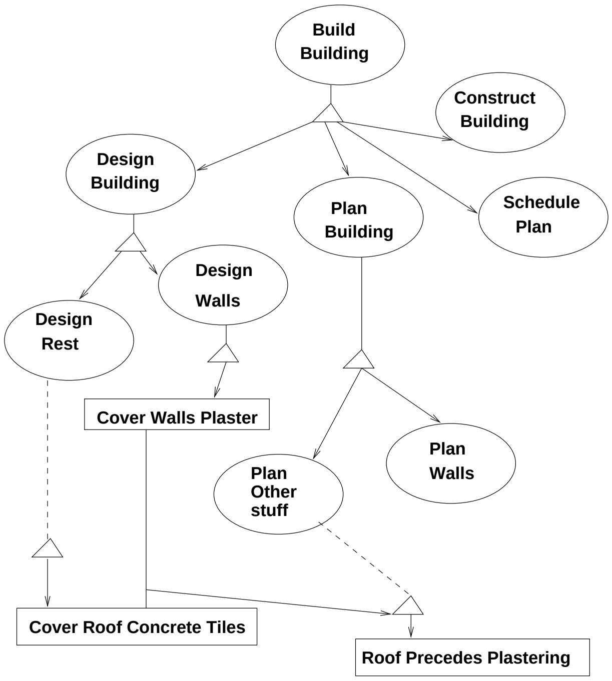

Figure 7: Complex Planning

This general idea can be elaborated in many different ways but it allows Redux to be used to manage the satisfaction of different aspects of the work, such as design and construction planning and scheduling. It requires only the general idea of a Plan Manager (PM) agent that either reviews the design periodically or requests Redux to be notified when design decisions are initially made. The Plan Manager then decides what aspects of the design to plan when, recording these decisions in Redux, using decision rationales and dependents to connect to the design features. An example of this general idea is shown in Figure 7.

4.2 Planning Design

The longer paper details further extensions to the Redux model required for tracking planning, scheduling, and execution.

Here we note only that the *project management representation presented here can be recursively applied to design itself.* Often, before any design is done, there is some design planning and scheduling. E.g., before a new CDROM player is designed, one plans for the design of the “actuator” as that is a common part. And one schedules the actuator design on the Gantt chart shown to management. Thus is the design process planned and scheduled prior to design. Since the actual design will at least add information to such planning and scheduling activities, if not correct them, it is necessary to use a mechanism such as the one presented in this paper to manage the design.

Thus we now have as a minimum, interactions between the components of

- the artifact design plan and schedule,
- the artifact design,
- the artifact plan and schedule of construction, and
- artifact construction.

Each includes design decisions that also interact with each other, also managed by the Redux model. Notice also that the design is the execution of the design plan and schedule, just as the artifact construction is the execution of the artifact plan and schedule of construction.

5 Summary

We have presented here a novel approach for managing complex distributed projects using agents and a particular representation based upon a general model of design: Redux. We have shown how design, planning, scheduling, and construction can be interleaved and distributed but coordinated by using a central facilitating agent, but which is not a planner or scheduler itself. Indeed, the design, planning, and scheduling for the project may be distributed as required among human and software agents as appropriate.

This enables, in principle, the management of much more complex projects then are now possible. Alternatively, it enables distributed projects to be completed more quickly, or to start earlier with less complete information because there change notification is supported.

This model has been modified for various applications. For example in CoMoKit[Dellen, 1997], invalid goals invalidate their decisions also. But the general model has proved useful and the first implementation of a Project Management system, Procura[Goldmann, 1996] was shown to work.

This paper is intended to assist other system builders with reusable principles in hopes that experimentation with this particular model of agent coordination will be accelerated.

6 Acknowledgments

Professor Mark R. Cutkosky of the Stanford Mechanical Engineering Department and and Professor Martin Fischer of the Stanford Civil Engineering Department have been of great assistance in developing and exploring these ideas. Stanford Center for Design Research work was funded by Navy contract SHARE N00014-92-J-1833 under the US DARPA RaDEO program.

7 WWW URLs

{0} http://cdr.stanford.edu/ProcessLink/
{1}
http://www-leland.stanford.edu/group/CIFE/VDT/
{2} http://ccs.mit.edu/pifintro.html
{3}http://www.sap.com/
{4} http://www.peoplesoft.com/
{5} http://www.digit.com/ap.html
{6} http:// KQML
{7} http:// FIPA ACL
{8} http://www.sics.se/isl/coord/
{9}
http://cdr.stanford.edu/ProcessLink/papers/white-dpm.html
{10} http://cdr.stanford.edu/ProcessLink/ABE/
{11}
http://cdr.stanford.edu/ProcessLink/talks/wulkow/proj-27.html
{12}http://cdr.stanford.edu/ProcessLink/talks/wulkow/proj-22.html
{13}
http://cdr.stanford.edu/ProcessLink/protocol/EPL-syntax.html

References

[Benda, 1998] Benda, M. internal survey of Boeing managers, 1998.

[Conklin, 1988] Conklin, J. and M. Begeman, “gIBIS: A Hypertext Tool for Exploratory Policy Discussion,” Proceedings of CSCW ’88 (Computer Supported Cooperative Work), September 1988.

[Cutkosky *et al.*, 1993] Cutkosky, M., et al., “PACT An Experiment in Integrating Concurrent Engineering Systems,”, *IEEE Computer*, January, 1993.

[Dellen, 1997] Dellen, B., Maurer, F., and Pews, G., “ Knowledge-based techniques to increase the flexibility of workflow management,” *Data & Knowledge Engineering*, North-Holland, 1997. See also http://wwwagr.informatik.uni-kl.de/~comokit/.

[Feldman, 1980] Feldman, Allan M., *Welfare Economics and Social Choice Theory*, Kluwer, Boston, 1980.

[Goldmann, 1996] Goldmann, S., “Procura: A Project Management
Model of Concurrent Planning and Design,” *Proc. WETICE-96*, Stanford, CA., June, 1996. See also http://cdr.stanford.edu/ProcessLink/Procura/papers/procura.l

[Haddadi, 1996] Haddadi, A., *Communication and Cooperation in Agent-Systems: A Pragmatic Theory*, Springer Verlag, Lecture Notes in Computer Science, No. 1056, 1996 .

[Kuokka and Harada, 1995] Kuokka, D. and L. Harada, "A Communication Infrastructure for Concurrent Engineering," *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (AIEDAM), /bf 9, 1995.

[Labrou and Finin, 1997] Labrou, Y. and Finin, T., "A Proposal for a new KQML Specification," U. of Mayland CS and EE Dept. TR CS-97-03, February 1997. Also http://www.cs.umbc.edu/kqml/.

[McKinsey, 1997] *McKinsey Quarterly*, No. 1, 1997.

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

[Petrie *et al.*, 1995] Petrie, C., Webster, T., and Cutkosky, M., "Using Pareto Optimality to Coordinate Distributed Agents" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (AIEDAM), **9**, 269-281, 1995.

[Petrie *et al.*, 1996] Petrie, C., Jeon, H., and Cutkosky, M., "Combining Constraint Propagation and Backtracking for Distributed Engineering," *ECAI-96 Workshop on Non-Standard Constraint Processing*, Budapest, August, 1996, revised for *AAAI-97 Workshop on Constraints and Agents*, Providence, RI, July, 1997. See also http://cdr.stanford.edu/ProcessLink/papers/non-stan-const/non-stan-const.html.

[Petrie, 1996] Petrie, C., "Agent-Based Engineering, the Web, and Intelligence," *IEEE Expert*, December, 1996. See also http://cdr.stanford.edu/NextLink/Expert.html

[Raquet, 1999] Raquet, A., "Dynamic Project Management for distributed Processes," Diplom Thesis, Kaiserslautern, 1999.