ProcessLink Coordination of Distributed Engineering

Charles Petrie
Center for Design Research
Stanford University
Stanford, CA 94305-2232
petrie@stanford.edu

August 29, 1997

1 ProcessLink Goals

The goal of ProcessLink is to provide a technical infrastructure and methodology that would allow globally-distributed engineers and designers and their heterogeneous tools to work together in structured coordination[1].

Today, projects composed of engineers and specialists in different disciplines are coordinated through management techniques that are unchanged in the last hundred years. There is a management hierarchy that controls the project, with more detailed tasks and fewer people controlled at each decreasing level in the hierarchy. Managers meet to exchange progress and coordinate the activities of the people who report to them.

There has been some recent work in analyzing some of the various organizational models used as well as various coordination models and determining which might go together best[11]. There has been very good technology developed for remote sharing of information so that people can have virtual meetings and perform some work as if they were co-located.

Much work has also been done in providing support for processes that are routine: ones in which the actions are repetitive and contain few exceptions. However, such *workflow* technology, though useful for document processing, is not useful for novel design among distributed participants. Especially, workflow tools do not support change propagation.

For novel design, the state-of-the-art is exemplified by current CAD tools in which each picture is given an electronic signature block. Whenever the picture changes, email is sent to the addresses listed in the "signature block", regardless of whether the actual change affects all, or even one,of the addressees.

There has been little progress in identifying needed coordination models that cover a wide class of applications and could be used to structure the work so that heterogeneous software and multi-discipline engineers could be coordinated more automatically. Constraint Satisfaction is a one model that has been imposed[1] but as it is not a model of design, it is inadequate in several respects. It usually requires enumeration of all possible options prior to designtime, there is no notion of design rationale, and there is no support for dynamic subgoaling: task decomposition. Similarly, there are problems with trying to use other non-design-specific models for design, such as integer programming[5].

What is needed is a model of dynamic design decision making. One such model is Redux[15]. One feature of the Redux model is that it can be very simple and also elaborated extensively. At the core of this model is a mathematical and rigorous but simple notion of a *design decision*. In this notion, a

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

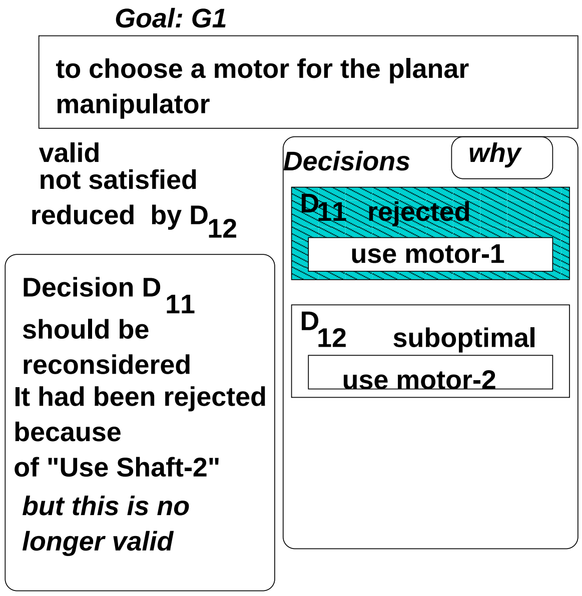

Figure 1: A Redux Decision

design decision must have at least two parts. The first part is called a *goal*: this is an issue, a task, a specification to be met, or any question that needs to be answered about the design. The second part is simply the answer to the question: the way to satisfy the specification or accomplish the task or resolve the issue. Answers can be of two kinds. An *assignment* is any statement about the design. And/or the goal may be decomposed into *subgoals* that when satisfied satisfy the initial goal.

Figure 1 shows an example of two Redux decisions in which the initial goal is "to choose a motor for the planar manipulator". Decision D_{11} was to use *motor-1* and Decision D_{12} was to use *motor-2*. The currently valid decision is D_{12} because previously D_{11} had to be rejected because of some conflict. But by indicating D_{12} is "suboptimal", in the sense of *pareto optimality*[6], Redux is saying that this conflict is no longer the case D_{11} should again be considered.

The basic Redux model is similar to that of gIbis[2]. 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 [16].

The ProcessLink project goal is to build on this model to implement an infrastructure that would provide some measure of automatic coordination. The basic idea is to use this common model to integrate people and software, perform change propagation, and notify exactly those people that are affected by the change, and tell them the effect of the change. The system should not itself make any changes to a design.

The dissiderata for this infrastructure is that it: 1) be applicable to existing "legacy" software, 2) be applicable to informal documentation, 3) impose a minimal structure on the process while providing a useful result, 4) reduce the requirement for a management infrastructure, 5) provide better information for managers, and 6) reduce the complexity of distributed projects so that they can proceed faster with better quality. Our target class of applications is electromechanical design.

The central hypothesis of the ProcessLink project is that Redux is a simple and ubiquitous model of design that is feasible to use in providing some automatic coordination of distributed engineering software and people. Because this model is simple and ubiquitous, it can retroactively be used to model existing software and systems. Because it infers the consequences of changes, the model is useful for coordination of distributed design projects.

In addition, we are using the technology of KQML-based agents[9] in order to provide a modular integration of distributed software and leverage Internet functionalities. In particular, we allow temporary JAVA-based agents to be created, and destroyed, at will by users, in addition to "wrappers" that convert existing software into agents. The technology and methodology we have developed for doing so, we call *Agent-Based Engineering (ABE).*

2 History of the Project

This project has grown out of a large body of work. There are three major threads. One important ancestor of ProcessLink was *First-Cut*, and its successor *Next-Cut*. These systems demonstrated the feasibility of providing some coordination of design and production. Another major thread was the agent work that resulted from the PACT[3] experiment and was used in the *First-Link*[12] project that showed how multiple-engineering disciplines could be incorporated into distributed agents that collectively performed design, in this case electrical cable harness design for aircraft.

The follow-on to this project was *Next-Link*, which had three major results. First, it showed that the Redux model, and the Redux' agent, could be adapted to existing systems. It demonstrated that the Redux' agent could provide increased coordination of the design. And Next-Link demonstrated how the agent infrastructure from the SHADE[8] project could be used to distribute the Next-Link agents anywhere on the Internet. Demonstrations of the various Next-Link CAD systems were routinely conducted with distributed agents, most running at Stanford, and one, Redux', running in Germany.

From this project, we learned a great deal that is being applied in ProcessLink. We have developed a more sophisticated model of coordination interactions and a KQML-based agent language to represent it. We have learned a methodology for adapting legacy systems And we have added additional generic agents, in addition to Redux'.

3 Organization of the Project

ProcessLink is one of three main subprojects under the larger *Design Space Colonization (DSC)* project sponsored by the DARPA RaDEO program. One of the other projects is it Process Open Description System (PODS) that is developing agents to allow designers to find, dynamically, information about manufacturing processes appropriate to their developing design, and then to download design constraints into the CAD system appropriate for that process. PODS and ProcessLink share a common agent infrastructure and can be used to support each other. For example, ProcessLink can be used to coordinate design decisions to use particular processes.

The third main subproject under DSC is DesignJournal, which is used to provide individual designers better tools to document their design and also to improve it. The connection with ProcessLink is that the Redux model is being used to structure, minimally, informal text so that some coordination can be achieved from distributed documentation. A major component of DesignJournal is *PENS*: an HTML authoring tool that facilitates sharing of informal notes among project participants.[2] A special version of PENS allows the user to add special tags by highlighting and clicking on icons. These annotations correspond to the Redux model. Agent messages are automatically sent to a Redux' agent that then sends coordination messages to applets on the users' desktop browsers when appropriate. In addition to DARPA, ProcessLink work is also sponsored by some industrial companies, most notably Toshiba, who also assist in providing industrial design applications.

[2] See also http://cdr.stanford.edu/DesignJournal/html/pens.html.

4 Products of the Project

The primary deliverable of ProcessLink is a coordination and integration framework that includes a sophisticated set of agent messages with process semantics, a core Redux' agent, and other generic agents. One of these that is best developed is a *Constraint Manager*[17] that provides generic constraint propagation services, integrates with the dependency-directed backtracking[14] services of the Redux' agent, and translates among external constraint solvers that may be "plugged-in" to the Constraint Manager. In figure 2, two domain solvers, one for discrete variables and one for continuous variables are shown. When domain solvers define constraints, they may define also value domains for each variable. This figure also illustrates that as problem solvers change these variable domains dynamically, the Redux agent tells the Constraint Manager. If necessary, the Constraint Manager then tells a particular domain agent that an answer to a previous request for consistent values is now different.

Another generic agent for which a first design and testable prototype has been developed is a *Project Manager*. The first design and prototype is called *Procura*[7]. The basic idea here is to use the notion that project planning and execution is also a type of design and extend the Redux model to include specialized project management concepts in order to provide highly flexible support for distributed projects in which the plan and the artifact being developed can change.

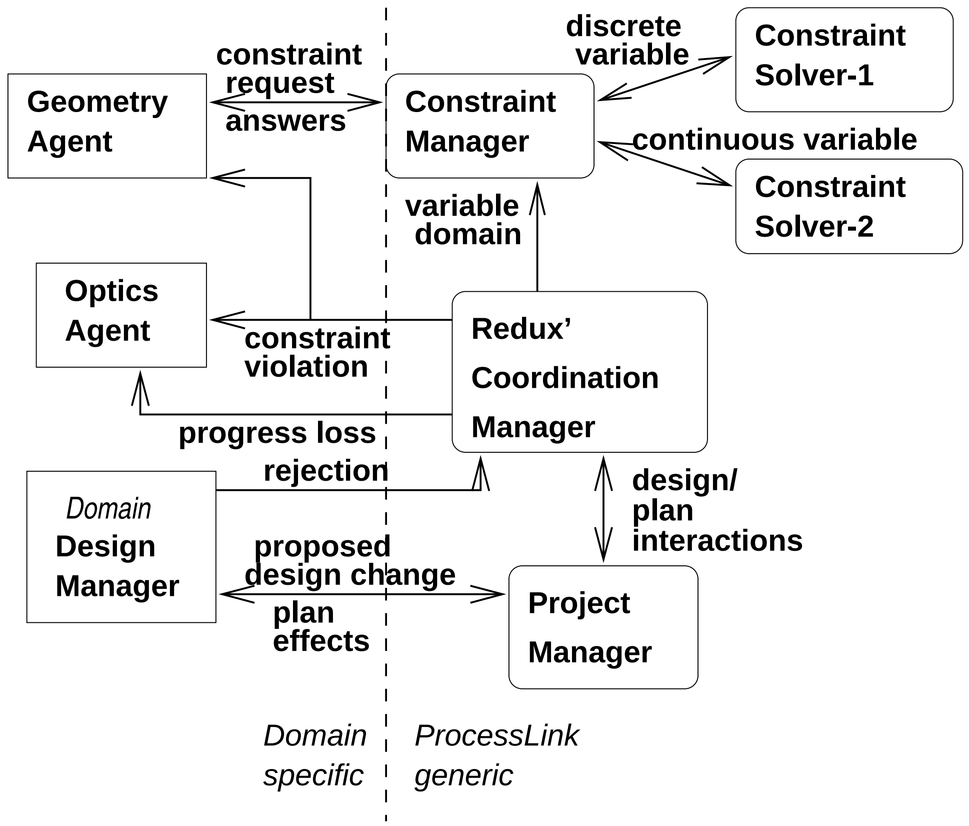

Figure 2: ProcessLink Architecture

Figure 2 shows how these components fit with example domain-specific agents in an architecture. The Constraint Manager, Redux', and Project Manager are domain-independent, generic agents. The messages being exchanged are examples of the domain-independent generic message types that are also part of the ProcessLink framework. On the left are domain-specific agents consisting of legacy software adapted to become agents able to send and receive such messages.

In this example, the Geometry Agent is specialized CAD software for drawing 3-D pictures and detecting geometric interference between features. This is the sort of software that would be used to, for example, design the frame for optics in a missile seeker or a compact disk pickup-head for a consumer product. The Optics Agent designs the distances and tolerances for the mirrors, objective lenses, and light detector for the same product. The two must evolve designs that are consistent and optimal - and as many designs should be explored as possible. Many other agents would in practice participate, including dynamic and thermal simulators. All of these specialized analysis and simulation software can be "wrapped" with code that converts them into agents with a common communication protocol.

A major deliverable of the ProcessLink project is an agent infrastructure. The current agent infrastructure developed for ProcessLink is called *JATLite*(or JATL)[3] and is available for use by other projects.

This infrastructure includes a JAVA-based agent router that will transfer and buffer agent messages so that they are not lost during a long-term project in which various software agents may be temporarily disconnected and other agents come and go. This is a practical but novel feature of our agent infrastructure. Most agent infrastructures do not provide this facility. Also, providing a message routing service, instead a simple nameserver, unloads the problem of each agent having to remember (and change) the IP address of all other agents with which it communicates. Additionally, the router models allows Java applets to act as agents because the router makes the socket connections connections to other agents, rather than the applet itself.

Also included is a JAVA-based template for constructing new agents, either completely new, or as "wrappers" to be added to existing software. Built-in to this agent infrastructure is support for common ways of communicating using the KQML agent protocol.

Figure 3 illustrates the use of the router for CAD software wrapped as agents, legacy process software wrapped as agents, and new stand-alone agents. The agent router in the middle transfers the messages and ensures the robustness necessary for support of long-term projects.

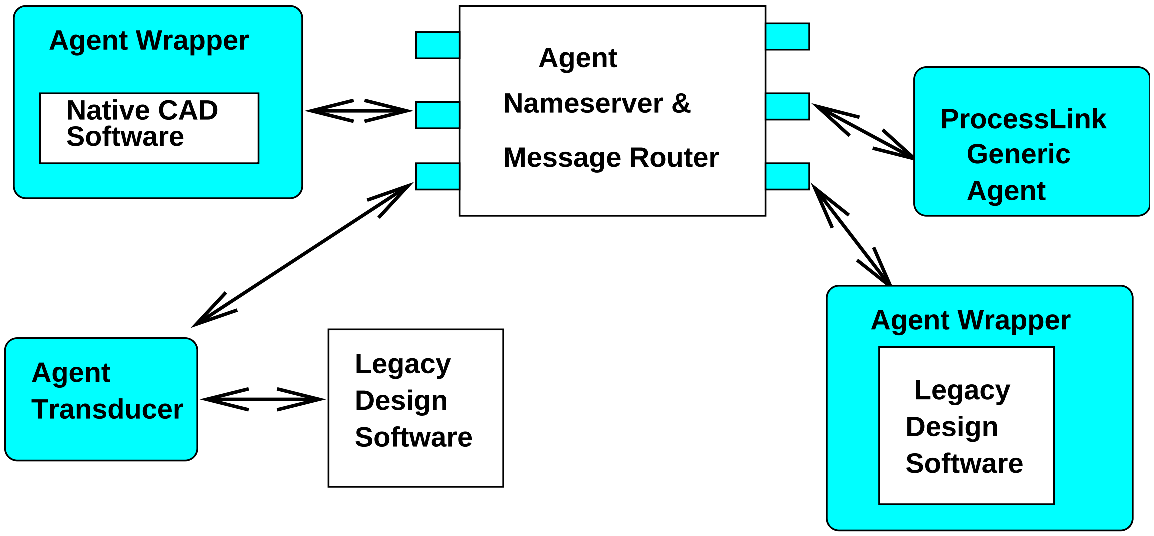

Figure 3: JATLite Infrastructure

The project also includes *DesignRoadmap*[13]: a new tool for analyzing and mapping processes. DesignRoadmap is used to perform an analysis of a design system or project in order to build the agents

[3]See also http://cdr.stanford.edu/ABE/.

that will support it using the ProcessLink framework. In particular, DesignRoadmap can be used to map the task inputs and outputs and this information is then used to build software that can be added to existing software so that it can send and receive agent messages. That is, the DesignRoadmap deliverable allows static analysis of the design process, prior to execution, which facilitates the design of the agent wrappers for each piece of design software. Built-in to each agent wrapper is some knowledge of what inputs and outputs are required by each agent, sufficient to carry out the whole design process.

The Redux model of design provides semantics that guide the final agent construction and determine the exact messages that will be exchanged among the agents. A Redux agent also listens "silently" to many of the messages and from time to time will offer designers advice on how better to coordinate the design. This advice is not based on any domain-specific knowledge but rather on a general model of design decisions and conflicts.

Finally, the new version of PENS, called *PENS-Rdx*, allows designers to send messages to Redux' simply by annotating text. Redux' does not understand the text itself, but the simple annotations denote the kind of object the text represents, and this is sufficient for Redux' to provide fundamental coordination services. The text annotation language is called *AEDNL*: An Electronic Design Notebook Language.

When designers enter design notes into the PENS server, they can be, if desired, shared with other designers as they are written in HTML and the PENS server is a web server. (PENS notes are communicated to the PENS server via email, however, so firewalls are not an issue.) When the notes are posted to the PENS server, a *proxy agent* reads the special AEDNL tags and converts the file into a KQML message that Redux will understand. Redux will occasionally return helpful advice to the designers.

Thus, each designer has on their desktop a PENS client into which they can write notes and annotate them with tags, and a JATL applet for communications with Redux and other agents. The Proxy agent always interprets the notes as messages being sent by an agent associated with the author and Redux always replies to that agent. If that agent is not currently online, the JATL router stores the messages. An example of a Redux-specific JATL applet can be found at http://java.stanford.edu/java_agent/jhc/JATLite/ReduxApplet.html. Such an applet can be used with or without PENS-Rdx and in combination with any other JATLite agent.

5 Evaluation/Experiments on the Products

We have experimented with various applications in collaboration with other universities as well as industrial companies. The software is still prototyped and we are constantly changing it as we experiment.

The initial experiments were with Next-Link on cable harness design. We have also worked with companies, such as Hughes Aircraft and Toshiba, on industrial electromechanical devices that are proprietary. One public reference is *Co-Mo-Kit*[4] that uses a re-implemented version of Redux for software engineering.

The current DARPA application is aircraft propulsion system design in conjunction with Rocketdyne and GE. In the future, we plan to offer an academic course in which students can experiment with ProcessLink technology.

6 Lessons Learned

A number of Redux functions have been modified or added as a result of these application experiments. The most valuable Redux coordination feature seems to be the identification of opportunities within a complex distributed design that result from conflicts and change. The basic idea is that if, say, two designers conflict in their design decisions, one may have to change their design and take a less preferable option. If later, the second designer changes the decision that had been in conflict, for whatever reason, the first designer should be notified that the initial more preferred option may now be available. A detailed scenario and explication of the underlying Redux technology is given in [16] and also identified as a fundamental coordination technology in [10].

6.1 Goal Assignment and Validity

Less well-known are other Redux functionalities, some of which have been developed during experiments. One that has always been present has been goal validity propagation. If a manager, for example, makes a design decision that to decompose a task into two subtasks, and assign them to coordinates, then if the manager changes the design decision later so that the subtasks are not necessary, then the subordinates should be notified. Redux does this automatically. Such a function is especially important if the subordinates themselves further decompose the subtasks.

In addition to this basic function, a new function has been added to allow the manager to state the assignment of the tasks. This causes Redux to automatically notify the subordinates that they have been assigned the task.

6.2 Input Change Notification

It has also shown to be useful to add to the notion of task *inputs* in Redux. It is important for a task "owner" to request to be notified when certain types of information become available. The notions of goal validity and assignment turn out to be integral with the notion of inputs.

To illustrate these concepts, suppose that a strategic-decision is made to use frameless motors in a device. Then one sub-task must be to develop frames for these motors as an integral part of the device. If framed motors are ever decided on instead, the person designing the overall device frame must be told, as well as the person evaluating the manufacturing process. This Redux did initially, by virtue of the goal/subgoal relation. However, we found that Redux must honor such requests only as long as the goal is valid. If the task of designing a frame is no longer valid, because of a higher-level decision to use framed motors, then the designer certainly no longer needs to be notified about the inputs for the now non-existent task. The Redux model permits this to be done automatically, without requiring the user to send explicitly a cancellation of the request.

6.3 Thrashing and Goal Blocks

In the context of electromechanical design, we have found instances of the utility of another Redux-based function - the prevention of thrashing within a design. When multiple designers conflict with each other and change their designs accordingly, it is possible for them to enter a repeating cycle unknowingly. Redux will advise engineers if they are at a point in the design where such a cycle is possible and will not allow entry into such a cycle without explicit manual override.

On the other hand, we are still not sure of the value of another Redux feature - *goal blocks* and their resolution. It is possible for one engineer to declare that a solution is not possible in obtaining a particular goal/ achieving a certain task. Redux will notify everyone who provides a possible input to this goal or task, as well as the designer who created the goal. They are notified that they must change specific design decisions in order for the blocked engineer to proceed. However, we have yet to observe use of this feature in practice. It may be that the novelty of the feature means that it will take practice to use it, or that the feature is simply not useful because no one is ever completely blocked in practice.

There are similar technical issues that we have learned, but a key one has been: *minimize messages to the user*. If the Redux' agent sends too many messages, they are ignored. We continue to work on eliminating some messages. For example, suppose that a designer decomposes a task into two subtasks, and works on those subtasks him/herself instead of assigning them to subordinates. Then Redux' suppresses notification that the supertask has been satisfied as soon as the subtasks are satisfied, because the user knows about this satisfaction already. In general, we work to ensure that the Redux' agent sends messages to the user that are in some sense surprising, rather than notifying the user of all changes in the design.

We have also found it good to redesign the KQML-based agent language in going from Next-Link to ProcessLink. Some of these changes are to include additional functionality and some for general functionality. For example, we have found it generally good for all messages to include a component that is computer-executable, such as a list of values, and a natural language text component that can be used, as a default, to present to the user, such as an explanation of the meaning of the list of values. Specialized application agents may generate special purpose text, but Redux always generates some general text with useful semantics.

7 Future Plans of the Project

In addition to finding more industrial applications for testing, we also intend to offer an academic class in which students would use ProcessLink and other design support tools. We also have identified several possible improvements and extensions.

One is the idea that designers do not simply make decisions about the design, but also about how to achieve information in order to make the best design decisions. These *metadecisions* should also be supported and we have had discussions with the *AG Richter* at the University of Kaiserslautern about how to do this. We are also working with them on future designs for project management, which we consider an important application.

In the very near future, we will extend the Constraint Manager to incorporate a variety of constraint solvers that were built elsewhere, and experiment with heuristics for integrating dependency-directed backtracking (DDB) with constraint propagation (CP). A key problem is that tractable CP algorithms do not guarantee consistency. However, given a constraint violation caused by use of CP, there is no explicit record of variable value rejections that could be used for DDB. There are very expensive algorithms that can be used to regenerate these rejections and their reasons. We are exploring the use of various heuristics to reduce this computational expense.

Finally, in order to reduce the number of messages to a user, we plan to build application-specific *message filtering agents*. Some messages from the Redux' agent are superfluous, but it requires specific domain knowledge to determine this. For instance, a Redux message may suggest that since the cost of a part has changed, and since the cost was "tagged" as part of the rationale for a design decision, then the decision should be reconsidered. However, it may be that the cost of the part is still less than any other part, or it may even have become cheaper. We believe such domain-specific reasoning should be added as additional agents to the framework. And the ProcessLink approach of agent-based engineering makes this relatively easy to do.

8 Acknowledgments

Professor Mark Cutkosky is the Principal Investigator of DSC and has been the lead researcher in this technology for several years. Neeraj Shodhan supports the ProcessLink infrastructure and the PENS-Rdx system. Heecheol Jeon is the creator of JATLite and the Constraint Manager and supports those systems.

Many other people have also contributed to this technology and are listed on the project web pages[4].

References

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

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

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

[4] 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/.

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

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

[7] 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.html.

[8] 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.

[9] 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/.

[10] Lander, S., "AI in Design: Issues in Multiagent Design Systems," *IEEE Expert*, April, 1997. See also http://computer.org/pubs/expert/1997/features/x2018/x2018.htm.

[11] Lee, J., and Malone, T., "Partially Shared Views: A scheme for communicating between groups using different type hierarchies," *ACM Transactions on Information Systems*, 8**(1)**, 1-26, 1990. See also http://ccs.mit.edu/pif1.html.

[12] H. Park, M. R. Cutkosky, A.B. Conru and S-H. Lee, "An Agent-Based Approach to Concurrent Cable Harness Design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 8, March, 1994, pp. 45-61.

[13] Park, H., "Modeling of Collaborative Design Processes for Agent-Assisted Product Design", Dissertation, Center for Design Research, Stanford U., January, 1995.

[14] Petrie, C., "Context Maintenance," Proc. AAAI-91, pp. 288-295, July, 1991.

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

[4] http://cdr.stanford.edu/ProcessLink/

[16] 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.

[17] 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.