Proceedings of DETC'98
1998 ASME Design Engineering Technical Conferences
September 13-16, 1998, Atlanta, Georgia, USA

DETC98/CIE-5697

AGENT-BASED COLLABORATIVE DESIGN OF PARTS IN ASSEMBLY

Toshiki Mori
Mechanical System Laboratory
Research and Development Center
Toshiba Corporation
Kawasaki, Japan
Email: mori@eml.rdc.toshiba.co.jp

Mark R. Cutkosky
Center for Design Research
Stanford University
Stanford, CA 94305-2232
Email: cutkosky@cdr.stanford.edu

ABSTRACT

In this paper, we propose an architecture in which engineering design agents interact with each other, exchange design information and keep track of state information to assist with collaborative design. We present an example involving CAD agents, for which each state corresponds to a particular design model. If a designer publishes a new design, the operation is recorded as a state transition that triggers action. Focusing on the history of design states and operations, we present a coordination algorithm that corresponds to the tracking of Pareto optimality. A prototype implementation is described, using a commercial 3D CAD system and agent interfaces written in Java.

INTRODUCTION

With increasing global competition, it is important to make all aspects of the design process as fast and efficient as possible. One way to achieve improvements in efficiency is through better coordination of the actions of designers working on a common design, so that unnecessary backtracking and delays are avoided.

By treating the collaborative design process as problem in agent interaction, it becomes possible to formalize the exchanges of information among designers and to bring formal tools to bear upon the problem of coordinating the actions of multiple designers. A particular motivation for the agent-based approach is the growing use of geographically distributed design teams that communicate over the Internet.

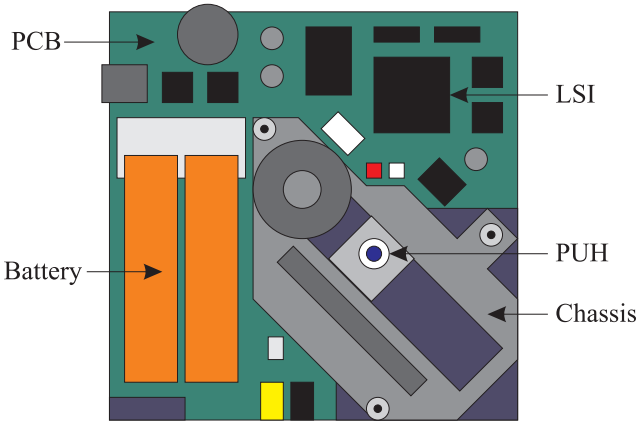

Figure 1. PORTABLE CD PLAYER.

Application domain

In this paper, we will examine steps in the design of a CD (compact disk) player. Figure 1 is a schematic representation of the inside of a portable CD player. All components, such as the pick up head (PUH), chassis, LSI chips and batteries, are packed into a compact case. Numerous engineers (mechanical designers, electrical designers, optical engineers, control engineers, heat analysts, vibration analysts, etc.) are required to develop the product. However, the design space for each engineer is so restricted that there are many conflicts between the engineers during the design process.

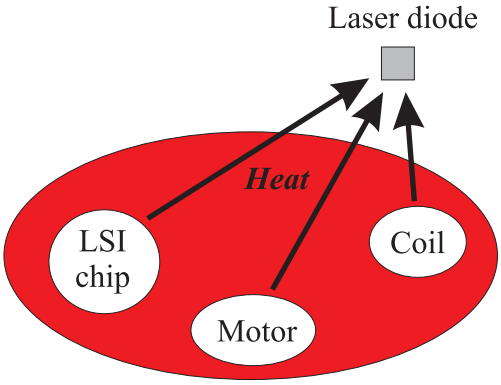

Figure 2. HEAT PROBLEM.

- *Geometric interference problem*

  One of most time consuming problems is layout of the CD player assembly, so as to achieve an efficient use of space while minimizing conflicts. Part of the problem is that different items in Figure 1 are under the control of different engineers. A chassis is designed by mechanical designers, whereas LSI chips are laid out by electrical designers. For efficient collaborative design it is important to catch conflicts and coordinate their resolution.

- *Heat problem*

  A second problem, related to the close packing of components, is the management of heat. In general, laser diodes both generate heat and are adversely affected by it. As shown in Figure 2, other heat sources include LSI chips, motors and coils. The configuration of these components, to minimize thermal problems while achieving a compact design, today requires a lengthy generate-and-test sequence, with extensive back-tracking.

RELATED RESEARCH

Agent-based Design Exploration

Bañares-Alcántara (1991) proposes an exploration-based model of design, describing the design process as a knowledge-based exploration task. In this view, design is classified as exploration, rather than search, because knowledge about the space of possible solutions has to be obtained before goals can be well formulated. The representation has a similar data structure to our model of the design process, using a state space where each state corresponds to a possible design. The representation is useful for design where the management of complexity and consistency plays an important role.

Malone et al. (1997) proposes two design principles for agent-based systems, through the experience of developing "intelligent agents" such as Information Lens (Malone et al. 1987) and Oval (Malone et al. 1995).

1. *semiformal systems*

   Don't build computational agents that try to solve complex problems all by themselves. Instead, build systems where the boundary between what the agents do and what the humans do is a flexible one.

2. *radical tailorability*

   Don't build agents that try to figure out for themselves things that humans could easily tell them. Instead, try to build systems that make it as easy as possible for humans to see and modify the same information and reasoning processes their agents are using.

The two principles imply that a certain amount of humility is desirable in proceeding toward the goal of building truly useful computational agents. Rule-based agents in our approach are suitable for those purposes. In fact, Information Lens and Oval are build on rule-based agents.

Agent-based Design Coordination

There is a growing body of work on agent-based coordination techniques for engineering design. In particular, *agent-based collaborative design* has been proposed as a promising approach for distributed engineering teams. The work in this paper is particularly inspired by two previous efforts which we briefly review here.

PACT (Cutkosky et al. 1993) is an experimental infrastructure for concurrent engineering. The architecture is based on interacting agents (programs that encapsulate existing engineering tools). The rationale for this approach is that individual engineering groups prefer to use their own tool suites and integration environments. By focusing on knowledge sharing through agent communication, the emphasis is on shared ontologies (shared concepts and terminology for communicating knowledge across disciplines) and language, rather than the exchange of data or objects. PACT demonstrated a distributed, multidisciplinary engineering simulation involving several academic and industrial partners. Agents communicated using the KQML (Finin et al. 1994) language and a shared ontology of terms and definitions, created expressly for the PACT exercise. However, the PACT participants also acknowledged that the creation of a shared ontology was a difficult and time-consuming process. Moreover, there was no particular mechanism for coordination of the agents in PACT. Agents broadcasted requests for information asynchronously and waited for other agents to provide an

answer.

A more structured approach to agent coordination is provided by a system called Redux (Petrie 1991), which is based on a TMS (truth maintenance system) (Doyle 1979) with dependency-directed backtracking. The Redux ontology characterizes design in terms of a hierarchy of *decisions*, *goals* and *assignments* of design variables, and the *rationale* that supports decisions. If two decisions are found to contradict, the model can be used to propagate appropriate state changes to affected model(s). In (Petrie et al. 1995) Redux is applied to an engineering framework for distributed cable harness design. The tracking of Pareto optimality is used to satisfy multiple objectives among distributed agents.

The Redux model is based on the process of general decision making, independent of a particular design. However, it requires a level of formality about announcing goals, decisions, assignments and rationale that can make it difficult to apply to engineering applications that are not already highly automated. In addition, there is some concern that a single Redux coordination agent could become a bottleneck in large distributed design projects.

DESIGN AGENT CONCEPT AND OPERATION

Our Approach

A practical alternative to the use of an external agent like Redux for design tracking and coordination is to distribute some of the functionality of Redux into each of the participating design agents in a collaboration. In our approach a design agent consists of a software tool, a wrapper that encapsulates the software tool to communicate with other agents, and a human designer who controls the agent itself. We do not yet believe that autonomous agents are practical for most engineering applications. Although our design agents rely on the same underlying coordination principle as Redux, i.e., the tracking of Pareto optimality, they do not use either TMS or the Redux model. Instead, they are equipped with a rule base that allows them to track dependencies and conflicts associated with mating 3D geometric models.

The design agents we propose are reactive agents that internally maintain states and rule-based knowledge. Each of the states is a design model at a particular stage in the design. If a designer publishes a new design model, the operation is recorded as a state transition that triggers actions according to the agents' rule-based knowledge.

Agent Operations

Figure 3 shows a representation of a design process as stored in an agent. It is based on the operator descriptions in STRIPS (Fikes and Nilsson 1971). Although STRIPS is a problem solver for robot planning, the representation is applicable for the recording and coordination of a design process. Each operation can be described as the set of three elements: (1) a *precondition* that must be true to apply the operation, (2) a *delete list* to be erased from the design model as the result of the operation, and (3) an *add list* to be added to the design model. In a particular domain where the set operations ($Union : \oplus$, $Subtract : \ominus$, $Intersect : \otimes$) are defined, such as solid modeling, *delete list* and *add list* can be represented as follows:

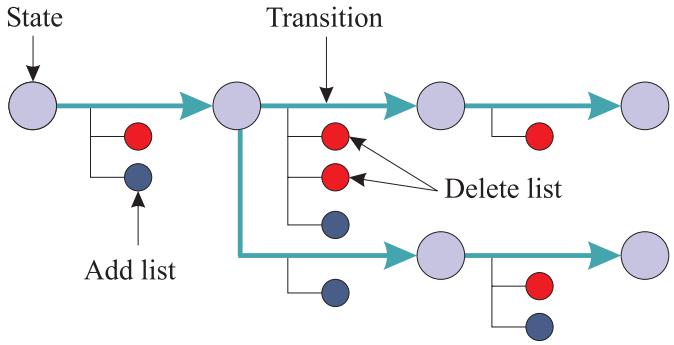

Figure 3. REPRESENTATION OF DESIGN PROCESS.

$$del\_list(T_j) := S_i \ominus S_j \tag{1}$$

$$add\_list(T_j) := S_j \ominus S_i, \tag{2}$$

where $T_j : S_i \rightarrow S_j$ is a state transition. Accordingly, the operation can be interpreted as the following rule:

$$\begin{array}{ll} \text{IF} & pre\_cond(T_j) \\ \text{THEN} & S_j \leftarrow S_i \ominus del\_list(T_j) \oplus add\_list(T_j). \end{array} \tag{3}$$

The designer's intention is involved in $pre\_cond(T_j)$. At the moment when the operation is executed, $pre\_cond(T_j)$ should be true. However, if $pre\_cond(T_j)$ subsequently becomes false, the operation might be no longer necessary. Therefore, the design agent automatically generates the following rule and keeps it as knowledge:

$$\begin{array}{ll} \text{IF} & \neg\, pre\_cond(T_j) \\ \text{THEN} & S_n \leftarrow Undo(S_k, T_j), \end{array} \tag{4}$$

where S_k is the current state and S_n is a new state generated. We can implement $Undo(S_k, T_j)$ with the command history management function of a solid modeling system. Otherwise, it can be defined as follows, on the assumption that any pair of operations are commutative:

$$Undo(S_k, T_j) := S_k \oplus del\_list(T_j) \ominus add\_list(T_j). \tag{5}$$

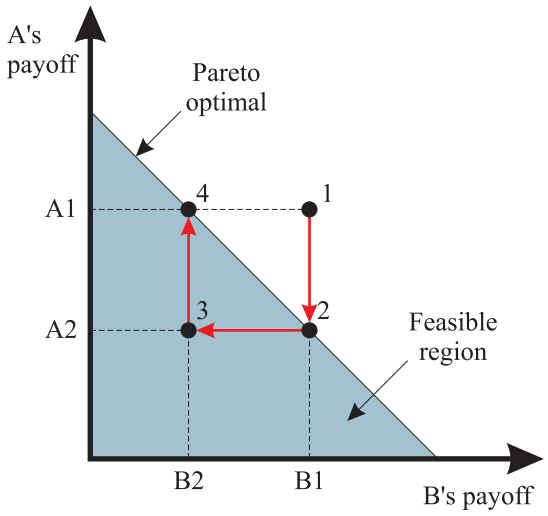

Figure 4. THE PAYOFFS OF PLAYERS.

This assumption is not necessarily valid in all domains, but it applies in many cases of practical interest. If necessary, we can add more rules that represent the dependency between operations.

COORDINATION ALGORITHM

Pareto optimality is a term of game theory (Fudenberg and Tirole 1991) for describing a solution for multiple objectives. An outcome of a game is *Pareto optimal* if and only if there is no other outcome that would give all players a higher payoff. Pareto optimality is an important property for distributed agents. In fact, Redux uses the tracking of Pareto optimality as a basic coordination function.

Figure 4 shows the payoffs for two players. Inside the feasible region (the region that satisfies constraints), players A and B can each increase their payoff without necessarily reducing the payoff of the other player. Conversely, outside of the region there is a conflict between the goals of A and B — either or both must accept a reduced payoff to bring the design back inside the feasible region. Pareto optimal solutions exist on the boundary of the feasible region. The numbered points in Figure 4 represent a sequence of designs and illustrate the tracking of Pareto optimality. At first, player A offers $A1$ and player B offers $B1$. Because they conflict with each other, player A reluctantly offers $A2$ instead of $A1$. Point 2 indicates a Pareto optimal solution. Let's suppose that player B offers $B2$ instead of $B1$ for some reason. Point 3 is not Pareto optimal — player A can now improve his payoff by returning to $A1$, without conflicts and without reducing B's payoff. The coordination that guides player A from point 3 to point 4 is called the tracking of Pareto optimality.

Focusing on the actions of agents, we can formalize a coordination algorithm that corresponds to the tracking of Pareto optimality. A "concessive action" is defined as one that makes a concession to avoid conflict. In a domain where set operations are defined, we can obtain a necessary condition for a concession to resolve a conflict or interference with another agent as:

$$del\_list(T_j) \otimes S_i' \neq \emptyset, \tag{6}$$

where S_i' is the state of the other agent at the moment T_j. We note that this is not a sufficient condition to eliminate interference. However, it provides a useful rule to incorporate into the knowledge base of each agent:

$$\begin{array}{ll} \text{IF} & del\_list(T_j) \otimes S_i' \neq \emptyset \\ \text{THEN} & T_j \text{ is a } concession. \end{array} \tag{7}$$

The concession is proper as long as $del\_list(T_j)$ interferes with the other agent. However, if the precondition becomes false, the concession might no longer be necessary. This logic can be represented as follows:

$$\begin{array}{ll} \text{IF} & T_j \text{ is a } concession \\ \text{THEN} & pre\_cond(T_j) \leftarrow del\_list(T_j) \otimes S_k' \neq \emptyset, \end{array} \tag{8}$$

where S_k' is the current state of the other agent.

The rules (7) and (8) are used as *meta-rules* to generate specific action rules for the coordination service. There are two advantages in this formulation. First, concessive actions of agents can automatically be detected by executing set operations — designers don't need to specify their intentions associated with design models. Second, the coordination can direct designers to newly generated states, in addition to backtracking to states previously visited.

Example

Figure 5 shows a simple example of a collaborative design process involving two agents, A and B.

1. Designer A creates a design model $A1$ and designer B creates a design model $B1$. There are two interferences between $A1$ and $B1$.

   $A1 \otimes B1 \neq \emptyset$ (Interference is detected.)

2. To eliminate the interferences, either designer A or designer B must modify the design model. In this case, designer A

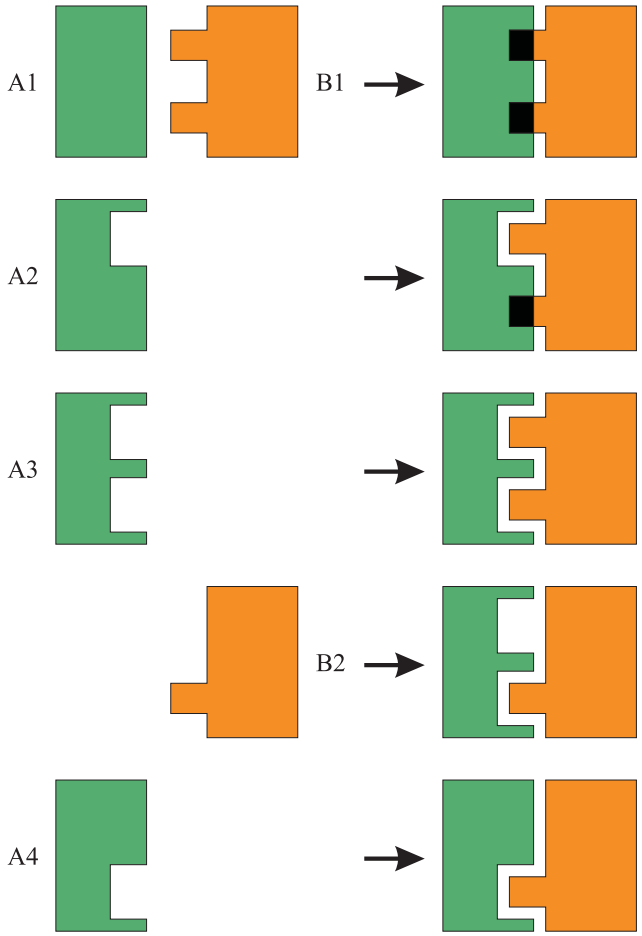

Figure 5. COORDINATION ALGORITHM.

makes a cavity on $A1$ and revises it as $A2$. There is still an interference between $A2$ and $B1$.

$T_{A2} : A1 \rightarrow A2$ (State transition occurs.)
$del\_list(T_{A2}) \otimes B1 \neq \emptyset$ (T_{A2} is a *concession*. [R1])
$A2 \otimes B1 \neq \emptyset$ (Interference is detected.)

3. To eliminate the interference, designer A makes a cavity on $A2$ and revises it as $A3$. Now, there is no interference between $A3$ and $B1$.

$T_{A3} : A2 \rightarrow A3$ (State transition occurs.)
$del\_list(T_{A3}) \otimes B1 \neq \emptyset$ (T_{A3} is a *concession*. [R2])

4. Designer B removes a boss from $B1$ and revises it as $B2$. A design agent assumed that designer A made a transition T_{A2} because of the boss. According to the tracking of Pareto optimality, the agent suggests that designer A cancel or undo the operation from $A1$ to $A2$. (In general, the transition could be made for some other reasons. Whether to cancel the operation or not will depend on the designer's choice.)

$T_{B2} : B1 \rightarrow B2$ (State transition occurs.)
$del\_list(T_{A2}) \otimes B2 = \emptyset$ (Rule [R1] is "fired".)
$A4 \leftarrow Undo(A3, T_{A2})$ ("Suggestion" is generated.)

5. Designer A accepts the suggestion from the agent and gets a modified model $A4$.

$T_{A4} : A3 \rightarrow A4$ (State transition occurs.)

The important point of this process is that designer A might miss the opportunity to cancel an unnecessary operation at step 5, without the coordination service. The result of the algorithm is a new design model $A4$ which is superior to $A3$ (assuming that designer A would like to take advantage of as much space as possible).

IMPLEMENTATION

We have implemented a prototype of design agents applicable to spatial interference problems with 3D-CAD systems. Figure 6 shows the architecture of the prototype. We selected AutoCAD R14[1] as an engineering software tool to build the design agents. ObjectARX[2] is the C++ API for the development of application programs with AutoCAD R14. The communication module is implemented with JATLite[3], an agent infrastructure designed to support the agents necessary for distributed design. It allows specialized JAVA applets to be downloaded on demand and work as agents, sending messages to each other and to stand-alone agents. The prototype uses a C++ version of JATLite to work with ObjectARX. The design process manager is a core module that records the design process, manages rule-based knowledge for coordination and controls the actions of the agents. Each of the design agents has a design process manager so that more than two agents can work together at a time.

The basic operations provided by the user interface of the agents are as follows:

Create is the operation that creates a new design model that has no ancestor.

Open is the operation that loads an old design model as the current model.

[1] AutoCAD is a trademark of Autodesk, Inc.
[2] ObjectARX is a trademark of Autodesk, Inc.
[3] Center for Design Research (Stanford). See http://java.stanford.edu/

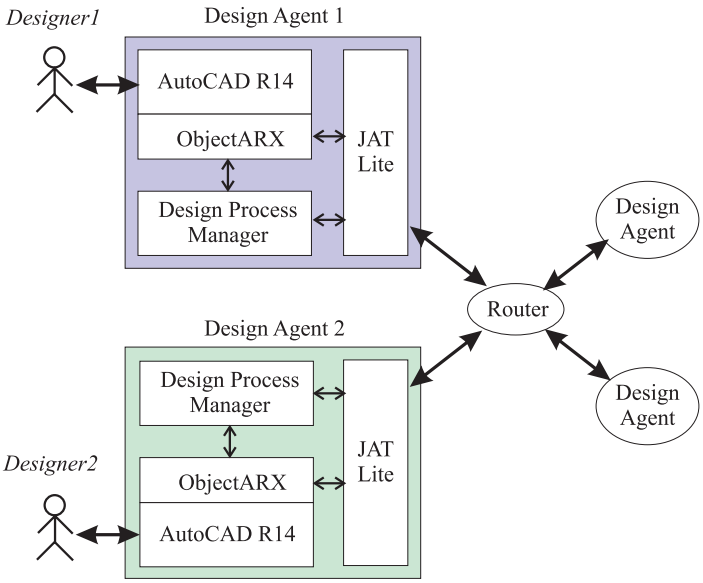

Figure 6. SYSTEM ARCHITECTURE.

Publish is the operation that saves a design model as a descendant of the current model, publishes a snapshot of the model onto the WWW (World Wide Web) and broadcasts the design model for other agents.

The user operations are very simple. The only thing designers have to do is to *publish* design models. This action may trigger a sequence of actions of related agents. The agents do not alter design models without permission of the designers. Instead, if the agents obtain some useful information, they post messages to the designers. Currently, we have two types of messages: *warnings* and *suggestions*. If a designer causes a conflict with another designer, the agent posts a *warning*; If a designer has an opportunity to make a design change, the agent posts a *suggestion*.

APPLICATION TO CD PLAYER DESIGN

To explore the utility of the CAD agents for CD player design, we have explored senarios involving designers working on an optical pick-up head assembly (PUH). Figure 7 shows a design scenario consisting of 8 steps. The PUH assembly includes a *bobbin* and a *base*. We assume that they are designed by different designers. In Figure 7, the first column shows a sequence of bobbin designs, the second column shows bases and the third column shows the assemblies. Note that the bobbins shown in the first column are enlarged with respect to the bases and assemblies.

1. The bobbin designer creates *bobbin1* and publishes it. The base designer creates *base1* and publishes it. The design agents detect two interferences between *bobbin1* and *base1* and send warnings to the designers. (*Bobbin1* interferes with the vertical section and the triangular mirror of *base1*.)
2. The base designer modifies *base1* into *base2* in an effort to eliminate the interferences and then publishes it. However, the design agents again detect an interference between *bobbin1* and *base2* and send warnings to the designers. (*Bobbin1* interferes with the vertical section of *base1*.)
3. The base designer then modifies *base2* into *base3* to try to eliminate the interference, and publishes it. The agents compute no interference between *bobbin1* and *base3*.
4. The bobbin designer decides to use a smaller objective lens. (This action could be an independent choice on the part of the bobbin designer, or a response to complaints from the base designer. The rationale does not matter for the purposes of the algorithm described in this paper.) The designer modifies *bobbin1* into *bobbin2* and then publishes it.
5. According to the diameter of the new objective lens, the bobbin designer modifies *bobbin2* into *bobbin3* and then publishes it. The base agent suggests that the base designer may cancel the operation from *base2* to *base3* because *base2* no longer interferes.
6. The base designer accepts the suggestion from the agent and uses *base2*.
7. According to the thickness of the new objective lens, the bobbin designer modifies *bobbin3* into *bobbin4* and then publishes it. The base agent now suggests that the base designer may cancel the operation from *base1* to *base2*.
8. The base designer accepts the suggestion from the agent and uses *base1*.

CONCLUSION

We propose an algorithm to support engineering design agents that interact through shared geometric models in a design space. The agents are wrappers for commercial CAD software and are reactive, in the sense that they track, and respond to, changes in the state of the design. Each of the states is a design model at a particular time. If a designer publishes a new design model, the operation is recorded as a state transition that triggers actions according to the agents' rule-base. Focusing on concessive actions (i.e., actions made to try to eliminate a conflict), we present a coordination algorithm that corresponds to the tracking of Pareto optimality. Two useful features of this algorithm are that concessive actions are tracked automatically and that the agents can direct the designers to new states (corresponding to new geometric models) not previously visited.

A prototype of the design agents has been implemented using a commercial 3D-CAD system and a previously developed library of programs that support agent communication over the Internet. To explore the utility of this approach for CD player design, we have developed design scenarios involving engineers

working on components of the pick-up head assembly.

In the future, we will extend the research in the following directions.

- Currently, we have a simple and limited communication protocol to exchange information between the design agents. To provide a more flexible service, we need to enrich the communication protocol of design agents to include contracts and negotiations.
- Currently, we assume that the positions of all components are given in terms of absolute coordinates with respect to a common reference frame to facilitate interference detection. However, it is easier to build an assembly model using relative coordinates. We should incorporate a spatial reasoner into the agent environment to accomplish coordinate transformations, and perhaps to perform other services such as generating swept paths, etc. This concept is similar to the "geometry server" used in Next-Cut (Cutkosky and Tenenbaum 1991) for agent-based process planning.
- As our major concern is a spacial interference problem, we utlize a solid modeling system rather than a generic software tool for detecting and tracking interferences. However, our basic approach for coordinating the actions of the designers is generic and can be applied in any context for which set operations apply.
- Finally, to establish the practical utility of the approach for working engineers, we must conduct tests of the agents in an industrial design setting.

ACKNOWLEDGMENT

We would like to thank Dr. Osamu Horigami, Dr. Koichi Ohtomi, Mr. Shinya Sekimoto, Dr. Charles Petrie, Mr. Heecheol Jeon, Dr. Kos Ishii and staff of Stanford University Center for Design Research (CDR), Manufacturing Modeling Laboratory (MML) and Toshiba R&D Center for their help and assistance. The work at CDR has supported by Toshiba Corporation and DARPA, under contract N00014-96-1-0679.

REFERENCES

Bañares-Alcántara, R., "Representing the engineering design process: Two hypotheses", *Computer-Aided Design (CAD)*, **23** (9): 595-603, 1991.

Cutkosky, M.R. and Tenenbaum, J.M., "Providing computational support for concurrent engineering", *The international Journal of Systems Automation: Research and Applications*, **1** (3): 239-261, 1991.

Cutkosky, M.R., Engelmore, R.S., Fikes, R.E., Genesereth, M.R., Gruber, T.R., William, S., Tenenbaum, J.M. and Weber, J.C., "PACT: An experiment in integrating concurrent engineering systems", *IEEE Computer*, **26** (1): 28-37, 1993.

Doyle, J., "A truth maintenance system", *Artificial Intelligence*, **12**: 232-272, 1979.

Fikes, R.E. and Nilsson, N.J., "STRIPS: A new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, **2**: 198-208, 1971.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S. and Beck, C., "Specification of the KQML Agent-Communication Language", The DARPA Knowledge Sharing Initiative External Interfaces Working Group, Feb. 9, 1994.

Fudenberg, D. and Tirole, J., "Game theory", The MIT Press, 1991.

Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A. and Cohen, M.D., "Intelligent information-sharing systems", *Communications of the ACM*, **30** (2): 390-402, 1987.

Malone, T.W., Lai, K.-Y. and Fry, C., "Experiments with OVAL: A radically tailorable tool for cooperative work", *ACM Transactions on Information Systems*, **13** (2): 177-205, 1995.

Malone, T.W., Lai, K. and Grant, K.R., "Agent for information sharing and coordination: A history and some reflections", *Software Agents*, edited by Bradshaw, J.M, AAAI Press, 109-143, 1997.

Petrie, C., "The Redux' server", *International Conference on Intelligent and Cooperative Information Systems (CoopIS)*, 134-143, 1991.

Petrie, C., Webster, T. and Cutkosky, M.R., "Using Pareto optimality to coordinate distributed agents", *AI/EDAM*, **9**: 269-281, 1995.

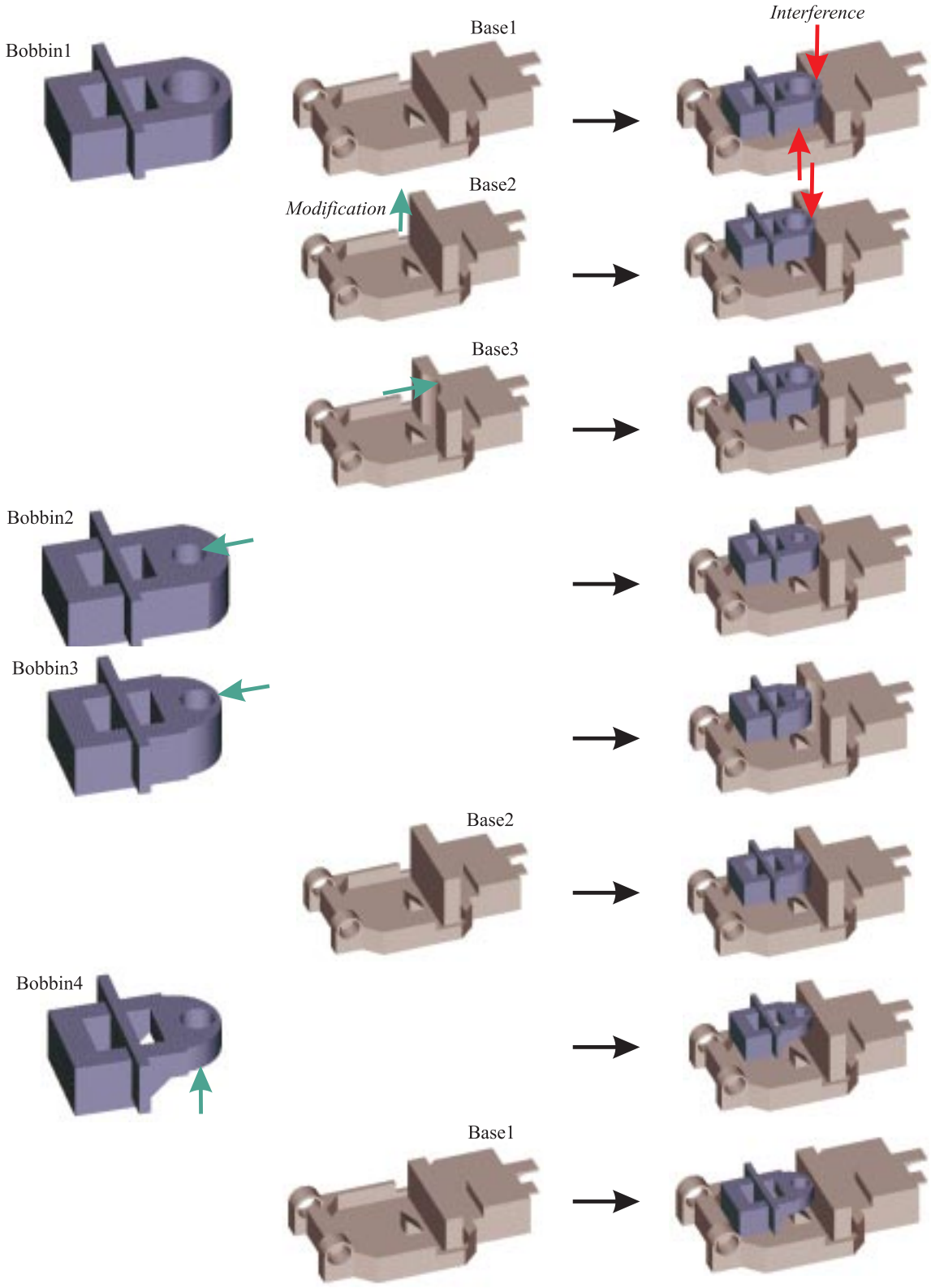

Figure 7. DESIGN SCENARIO OF PUH.