DIDE: A Multi-Agent Environment for Engineering Design

Weiming SHEN, Jean-Paul A BARTHES
CNRS URA 817 HEUDIASYC
Université de Technologie de Compiègne
BP 649, 60206 COMPIEGNE, FRANCE
E-mail: [wshen ; barthes]@hds.univ-compiegne.fr

Abstract

Real world engineering design projects require the cooperation of multidisciplinary design teams using sophisticated and powerful engineering tools. The individuals or the individual groups of the multidisciplinary design teams work in parallel and independently with the different engineering tools which are located on the different sites for often a long time. In order to ensure the coordination of the design activities of the different groups or the cooperation among the different engineering tools, it is necessary to develop an efficient distributed intelligent design environment. This paper discusses a distributed architecture for integrating such engineering tools in an *open* design environment organized as a population of asynchronous cognitive agents. Before introducing the general architecture and the communication protocol, issues about the agent architecture and the inter-agent communication are discussed. A prototype of such an environment with seven independent agents located in the different workstations and microcomputers is presented and an example of a small mechanical design is used for demonstrating such an environment.

1. Introduction

Real world engineering design projects require the cooperation of multidisciplinary design teams using several sophisticated and powerful engineering tools. The individuals or the individual groups of the multidisciplinary design teams work in parallel and independently often for quite a long time with different engineering tools which are located on the different sites. On the other hand, at any instant, individual members may be working on different versions of a design and viewing the design from various perspectives (e.g., electronics, manufacturing, planning), at various levels of details. In order to coordinate the design activities of the various groups and to guarantee a good cooperation among the different engineering tools, it is necessary to develop efficient distributed intelligent design environments. Such environments should not only automate individual tasks, in the manner of traditional computer-aided engineering tools, but also help individual members to share information and coordinate their actions as they explore alternatives in search of a globally optimal or near-optimal solution. A number of researchers have proposed to use distributed problem solving technology for concurrent design (Gero 1987; Morse 1990; Sriram et al 1991); or developed some agent-based systems such as PACT (Cutkosky et al 1993), First-Link (Park et al 1994), Next-Link (Petrie, Cutkosky, & Park 1994), Anarchy (Quadrel et al 1993), some multi-expert systems such as DICE (Sriram et al 1989; Sriram et al 1992), DESIGN-KIT (Stephanopoulos et al 1987; Sriram et al 1989), ANAXAGORE (Trousse 1993), CONDOR (Iffenecker 1994), EXPORT (Monceyron & Barthès 1992), ARCHIX (Thoraval & Mazas 1990) and a project at UTC (Ribeiro Gouveia & Barthès 93), some specific computer tools for inter-agent communication such as KQML (Finin, McKay, & Fritzson 1992), *ToolTalk*^{TM} (Frankel 1991), and (Populaire et al 1993), and some frameworks for inter-agent control (Van 1990; Lee, Mansfield, & Sheth 1993; Quadrel et al 1993; Boissier 1993).

We are currently developing a prototype of Distributed Intelligent Design Environment (DIDE) based on an architecture called OSACA (Open System for Asynchronous Cognitive Agents) (Scalabrin & Barthès 1993), derived from previous work in the domain of robotized assembly systems (Abriat 1991). Our goal is to verify whether it is actually possible to build truly open systems, that is, systems for which users can freely add or remove agents without having to halt or to reinitialize the work in progress in any way. We also want to exercise the obtained prototype in order to gain first hand experience, first with small examples, then with larger projects. Indeed, we are interested in very large design projects of complex systems such as an automobile, a locomotive, a harbor, or an aircraft. This type of design projects have the same characteristics: the design requires a large number of individual designers or several working groups to work together, the design period is long, and the design is very expensive. In this case, cooperation and coordination are very important, and the asynchronous cognitive agents are adaptable. In particular, in this context, we like to compare experimentally an agent-based approach with a more conventional blackboard approach like for example the one we already developed for harbor design (Monceyron & Barthès 1992).

The paper is organized as follows: Section 2 discusses the internal structure of a single agent, as well as the inter-agent communication. Section 3 presents the general architecture of DIDE. Section 4 describes an implementation with a small mechanical design example. And finally section 5 gives some concluding remarks.

2. Agent Architecture and Inter-Agent Communication

2.1. Internal Structure of an Agent

In our distributed intelligent design environment, agents are autonomous cognitive entities, with deductive, storage, and communication capabilities. Autonomous in our case means that an agent can function independently from any other agents. Fig 1 shows the internal structure of an agent (Ribeiro Gouveia & Barthès 1993).

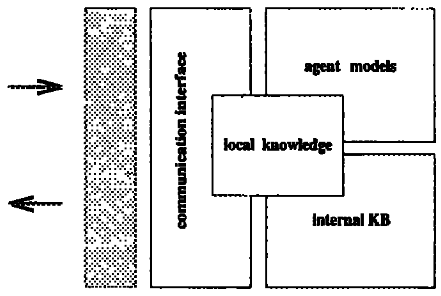


Fig 1. Internal structure of an agent

An agent is composed of: (i) a network interface (shaded portion in Fig 1); (ii) a communication interface, which provides handlers for I/O notification, or exceptions; (iii) symbolic models of the other agents, and associated methods to use them; (iv) a model of its own expertise which is application dependent; (v) a model of the task to be performed, or of the current context.

When an agent is connected to a group of active agents, then only its communication interface and its own expertise contain information. The part recording facts about the work to be done, or the capabilities of the other agents is empty. Each agent builds its own image of both the work to be done and the capabilities of the other agents on the fly, by processing the various messages it receives or exchanges. Of course, there are some difficulties when doing that. One of them concerns sharing the names of external objects when sensors are involved (Abriat 1991).

Several strategies can be used to transfer some knowledge about the work to be done or about the capabilities of other agents. A first one is to be passive, letting the agent ask questions whenever needed while executing a particular task. A second one is to have "curious" agents that have some knowledge of their ignorance, and thus are capable of asking some questions to improve their knowledge of the problem to solve. A third one would be to have some agent that teaches newcomers (agents) about the task being performed and the capabilities of the existing agent team. So far, in our experimental environment, we only considered the first and last hypotheses.

A cognitive agent is an agent which has at least the following properties: (i) it is knowledge-based; (ii) it has knowledge about other agents and the knowledge is obtained during the interaction or communication with other agents or learned from a special "tutor" agent.

2.2. Inter-Agent Communication

There are two models of inter-agent communications (Monceyron & Barthès 1992). The first one, used in blackboard architectures, consists of sharing information, i.e., the current solution of the problem is stored in a global common data structure, and is shared by all agents in the system. In design terms, the sharing or exchanging of information may be relative to the artefact or to the control of the solution and its management. The initial data and the different partial results which represent the successive versions of the artefact are stored in this kind of data structure. It is the only means for exchanging information. The ITX system (Lee, Mansfield, & Sheth 1993) has used this model of communication for a teleconferencing environment. This model of communication was also used in some projects for engineering design, such as DICE (Sriram et al 1992), DESIGN-KIT (Stephanopoulos et al 1987), ANAXAGORE (Trousse 1993), CONDOR (Iffenecker 1994), EXPORT (Monceyron & Barthès 1992), and ARCHIX (Thoraval & Mazas 1990). The second model of communication consists of transferring information, i.e., each agent builds its own model of the current solution by acquiring information from the other agents. This model of communication needs a protocol and a format for the messages (common message language) for expressing requests and replies. Each agent stores the current solution, or at least part of the solution, in its local database. Some projects such as MARS (Abriat 1991), PACT (Cutkosky et al 1993), First-Link (Park et al 1994) and Next-Link (Petrie, Cutkosky, & Park 1994) have used this model of communication. It is also used in our distributed intelligent design environment (DIDE).

Communication among agents in DIDE is flexible, asynchronous, and of multicast type. An asynchronous multi-agent communication is adaptable to a distributed intelligent design environment for large design projects. Indeed, even if synchronous communications (blackboard style) are very useful for systems in which the cooperating agents have to work simultaneously, such as a multimedia teleconferencing system

(Lee, Mansfield, & Sheth 1993), they are not required for large design problems, for which the design period is often very long.

One of the major problems for developing truly open systems is that of being able to insert or to remove agents on a given application without halting or reinitializing the (distributed) system. Indeed, since large design projects last several years, new tools or new services appear during the life of the project; similarly, existing tools get upgraded. Thus, it is necessary to accommodate such changes smoothly without disturbing the project. Traditional approaches, in which a group of powerful tools may be integrated into a large, efficient, decision support system, do not allow it. Such an approach is viable only if the problem domain is static, i.e., the tools, design rules, and production process do not change over the product life-cycle. Indeed, when new services are appended to a group of cooperating processes it is usually necessary to recompile all programs on all machines on the network. This is clearly unacceptable. New tools are becoming available that allow to avoid such a problem, although they are currently far from satisfactory.

Finally, on the communication side, the message content is of a fairly high semantic level. For example:

Open breakwater-entrance by 10 degrees

This raises the question of shared vocabulary and common ontologies, and also of the initial expert knowledge content of each agent prior to its connection to the network. A review of such questions can be found in (Cavalli et al 1991) in the context of "Enterprise Integration" and in (Gruber 1993) in the context of "Ontologies and knowledge sharing." Our approach in this domain is to manually organize needed concepts into minimal ontologies as needed.

3. A General Architecture for Design Environment

OSACA (Open Systems for Asynchronous Cognitive Agents) is a general architecture that we apply to our distributed intelligent design environment (DIDE). DIDE is a programming environment under development, in which the techniques of distributed artificial intelligence are used to realize the interaction and communication among multiple cognitive agents. Such agents may be connected with existing engineering tools or database/knowledge base systems, or may also be connected with some user interface for human domain specialists. Two key issues are how to select a way of communication (protocol) among such agents, and how to organize each tool associated with a cognitive agent.

We assume that all agents are connected by means of a network - local network or Internet. Each agent can reach any other active agent by means of a broadcasting message. All agents receive messages. They may or may not understand such messages. When they do not understand a message, they simply do nothing. Whenever they understand the message, then they start working on it, provided the priority of the message is higher than the current work they were doing. Thus, agents are multi-threaded. When a new agent is introduced, it is simply connected to the network. From then on, it receives messages like the other agents.

In our design environment, an agent offers some specific service, usually by encapsulating an engineering tool. The agent interaction relies on three things (Cutkosky et al 1993): shared concepts and terminology for communicating knowledge across disciplines, an interlingua for transferring knowledge among agents, and a communication and control language that enables agents to request information and services. This technology allows agents working on different aspects of a design to interact at the knowledge level, sharing and exchanging information about the design independently of the format in which the information is encoded internally.

Task Structure

Task execution is initiated locally and can be done in different manners. Firstly, agents can broadcast information to all the other agents and then wait until some agent has computed the answer. In this way the task can be carried out by several specialists of the subject working in parallel (provided they reside on distinct machines). Secondly, a contract protocol can be used, i.e., an agent broadcasts an offer describing the job to be done, waits for some time for submissions from the other agents, and then awards the contract to a particular agent according to some local criteria. Thirdly, a contract can be allocated directly to a known specialist. Note that although the last solution could appear to be the most efficient, a general broadcast allows all the agents to see the content of the request. Therefore, even if agents are not directly concerned by the request, they can nevertheless use part of its content to update their internal representation of the task being done or of their image of the expertise of the requesting agent.

4. Implementation

We have built an experimental distributed intelligent design environment (DIDE) on a network of SUN and VAX workstations and microcomputers. Each agent is implemented as a MOSS environment (Barthès 1989), a system of recursive frames capable of modeling objects with versions and well adapted to design activities. MOSS is itself constructed on top of Common LISP. Agents are built as a set of MOSS objects together with their behavior. Communications are implemented by using *ToolTalk*^{TM} for local communication and employing the ELM mail system for remote communication among the agents located in the different local networks. Communications must be programmed using C++.

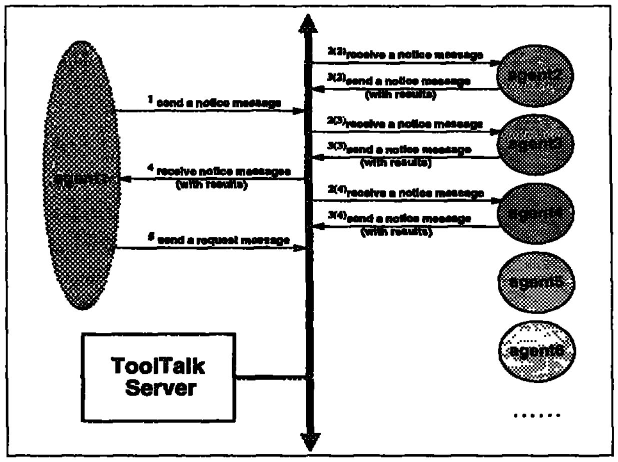
4.1. ToolTalk for Local Communication

ToolTalk^{TM} was developed by *SunSoft*^{TM} as a part of *Solaris*^{TM} *OpenWindows*^{TM} (Frankel 1991). Programs interact with *ToolTalk*^{TM} by calling functions defined in the application programming interface (API). The API gives *ToolTalk*^{TM} the appearance of a simple library, hiding the facts of its implementation as both a library and a collection of communicating processes running on computers throughout a local area network. *ToolTalk*^{TM} allows processes to communicate in a variety of ways with minimal knowledge of each other and no knowledge of the local area network. It acts as a message switch, tracking automatically the various process identities and locations. *ToolTalk*^{TM} makes use of the local network as a "software bus" into which engineering tools may be freely plugged and unplugged by the user. Connected tools automatically register themselves to send and receive messages that they can understand and act upon.

ToolTalk^{TM} thus gives us the capacity of connecting our various agents located anywhere on the local network without having to keep track of where they actually reside. In addition, such connections can be dynamic.

Communications with *ToolTalk*^{TM} are done using two types of messages: notices and requests (corresponding to publish and request in First-Link (Park et al 1994) and Next-Link (Petrie, Cutkosky, & Park 1994). A notice message is informational – a way for a process to announce an event. Processes that receive a notice absorb the message without returning anything to the sender. A request message is a call for action, the results of which must be returned to the sender as a reply. This is a too simple approach for our purposes. However, we implemented the various communication modes by using mainly notice messages (which contain the identity of the sender). A typical mechanism for the message management is shown as in Fig 2.

This paragraph shows how one can use the *ToolTalk*^{TM} notice and request message types to implement a contract protocol. Consider the six agents (corresponding to six engineering tools) of Fig 2. Each agent only knows *ToolTalk*^{TM} and a set of patterns identifying the type of profile defining another agent. Agent 1 sends a notice message to *ToolTalk*^{TM} describing a job to be done. Agents 2, 3 and 4, having a pattern that matches the pattern of the notice message sent by agent 1, process the message in parallel. Then, they each return a notice message with the results to *ToolTalk*^{TM}. Agent 1 receives all of these three notice messages sent by agent 2, 3 and 4. Agent 1 then analyses and compares the offers from the three different engineering tools, and can make a decision as to which tool to be selected for performing the required job. Agent 1 then sends a request message (point-to-point) for granting the job to the selected agent (selected tool). The final result will be automatically returned to Agent 1.



2(2)receive a notice message
3(2)send a notice message (with results)
agent2
1 send a notice message
2(3)receive a notice message
3(3)send a notice message (with results)
agent3
4 receive notice messages (with results)
agent1
2(4)receive a notice message
3(4)send a notice message (with results)
agent4
5 send a request message
agent5
agent6
ToolTalk Server
......

Fig 2. A mechanism for message management under the ToolTalk Service

In the agent, the message is described as a MOSS object, and must be transformed into a message attribute list and a string (including other data or results) before it can be sent.

Although *ToolTalk*^{TM} suffers from a number of problems, it was nevertheless available, which is the main reason for using it. We are developing other protocols and tools for the OSACA multi-agent platform.

4.2. ELM for Remote Communication

In large design projects, the various activities and their corresponding computational tools may be located on different sites, and may not be on the same local network. In this case, ToolTalk falls short from meeting the requirements.

Electronic Mail (E-Mail) services as the primary medium for both human communication and tool communication. Engineers use e-mail routinely for discussions and information exchanges. Computer agents can do the same. Using structured messages, they can request information and services from other agents, communicate results, notify agents of design changes, and so forth. E-mail has a combination of characteristics that make it well-suited for integrating loosely coupled applications in wide-area networks demonstrably heterogeneous and scalable. Secondly, recent standards in multimedia e-mail make it possible to exchange compound documents containing text, images, audio, video, and programs. Thirdly, it is relatively easy to connect existing applications to e-mail. Finally, e-mail's explicit asynchrony matches the way engineers prefer to work. Indeed, publishing design changes and "checking-in" for changing notices at will allows them

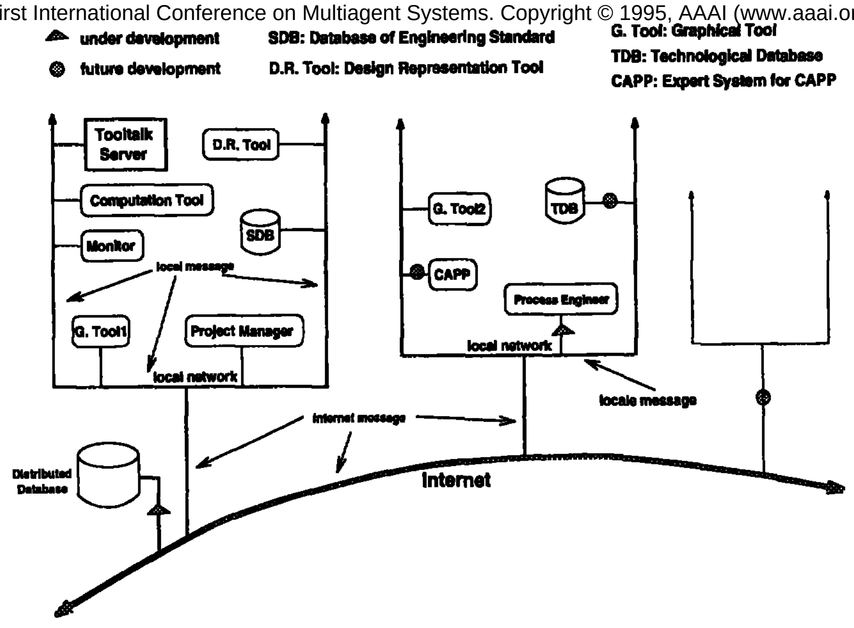


Fig 3. An Experimental System Architecture of DIDE

to feel in full control of the design process (Toye et al 1993).

In DIDE, an e-mail tool called ELM is used for remote communication, i.e., for exchanging information among the agents located in the different local networks.

4.3. Version Mechanism for Conflict Resolution

Conflicts occur in multidisciplinary design environments mainly for two reasons: individual participants in the cooperative process lack the complete knowledge of global objectives necessary to ensure that their locally-preferred solutions are in alignment with these higher objectives, and individual disciplines have individual ideas about what constitutes the best design. Even individual design requires trade-offs because of competing design criteria, such as safety, cost and social acceptance, as well as artifact requirements and specifications. The ability of designers to avoid or minimize conflicts through judicious trade-offs and other methods is one of their most valuable skills.

Resolution and detection of conflicts are especially difficult when the design task as well as knowledge concerning such competing factors are distributed among different actors with different perspectives. Certain methods such as negotiation, hierarchical structures, constraint weakening, creation of new variables and user interaction can be used for conflict resolution. It is also possible to combine several methods in the same system.

In DIDE, we leave each group of designers develop possible conflicting partial solutions (divergence) (Barthès 1993). Such solutions are managed as parallel versions. Then, at the review meetings all groups have to compromise on a commonly agreed solution or on a solution imposed by the project manager (reconciliation). This relates to the global consistency of the project and not to the consistency within each partial solution, which we have not yet addressed (see also *Project Manager* in the next section).

4.4. A Small Experimental Example for DIDE

DIDE currently comprises seven independent agents: project manager interface, design representation, engineering calculation, database of national engineering standards, agent administration monitor, a 3D graphical environment (AutoCAD on SUN/UNIX) for displaying the design results, and another 3D graphical environment (SDRC I-DEAS on VAX/VMS) for process engineers (as shown in Fig 3):

Project Manager: The interaction mechanism for agents in DIDE has been developed to meet the needs of human designers working on a collaborative project. Here the agent *Project Manager* is taken as this type of "human" agent for specifying the initial parameters and some constraints for the design project. Its potential functions which are under development include the version control for conflict detection and resolution by using the strategy of divergence/reconciliation (Barthès 1993) and by combining two methods: user interaction and the optimum calculation for some special constraint conflicts. In the future DIDE environment, there will be several agents for human specialists in the different domains such as dynamics, electronics, economics and process engineering.

Design Representation Tool: This agent is responsible for the generation and management of geometric entities and their properties of a particular mechanical system. The geometric information imports from or exports to a specific CAD tool via the agent *Graphical Tool1*. All classes, instances and methods are defined in this tool. The objects can be stored in an object-oriented database (e.g., $MATISSE^{TM}$). The propagation of modifications (Shen, Barthès, & EL Dahshan 1994) is done by this agent.

Computation Tool: This agent is responsible for engineering calculations. It may be a Finite Element Analysis tool or some other engineering computation tool. Currently it is simply used for bolt stress calculations.

SDB (Database of Engineering Standards): A database of national engineering standards is very useful in the domain of mechanical CAD. This agent is served for searching required standard dimensions in the database of engineering standards.

Monitor: This can be seen as an administrative agent. Generally it works for the project manager. It receives all messages sent by all the other active agents in the same system and displays some important information on a specific interface. It stores all information about the task being performed and the capabilities of all other agents in its temporal local knowledge base. When a new agent connects to the system or an active agent withdraws, it will update its knowledge base and send a broadcast message to all other agents. It can also serve as a "tutor" for newcomers. (This is however not the case as of now).

Graphical Tool1: Currently this graphical tool is AutoCAD, located in a microcomputer, which is used to display the design results on a 3D graphical interface. It can be also used to prepare a mechanical drawing or to modify the drawings by a human domain specialist, in the latter case, the agent is taken as a human agent. This tool will be soon replaced with a UNIX version of AutoCAD on a SUN workstation, which will facilitate the storage and retrieval of the graphical objects. The data exchange with other agents is done by means of standard DXF (Drawing Interchange Format) files.

Graphical Tool2: This graphical tool is SDRC I-DEAS located on the network of VAX workstations. This graphical interface allows process engineers to verify technological constraints. This agent communicates with other agents by means of the ELM mail system.

We illustrate how easily each agent can communicate with other agents using a small academic example. We consider a mechanical assembly as a design example, already described in details in (Shen, Barthès, & EL Dahshan 1994). If a designer (here the project manager) desires to design or redesign this mechanical assembly, he can specify all the necessary parameters such as the total pulling force and some constraints such as the bolt head type, the safety factor by means of a specific interface for agent *Project Manager*. The agent *Project Manager* sends a notice message with a predefined pattern and other initial data. The agent *Computation Tool* receives this message by matching the pattern and takes the required calculation to obtain a bolt diameter as a result, and then sends a notice message with this result and a predefined pattern. The agent *Design Representation Tool* receives the message by matching the pattern and begins to design or modify the mechanical assembly. When it needs standard tables for obtaining the standard (nominal) diameter of the bolt or the nut or the washer, it sends a notice message with a predefined pattern and calculated data and waits for a reply. Here we use notice message but not request message because there may be several databases of engineering standards in the future design environment. The agent *SDB* receives this message by matching the pattern and searches required information in the database of national engineering standards and then sends a notice message with the results and a predefined pattern. The agent *Design Representation Tool* receives this message and continues its design or modification. After the design or modification is finished, the agent *Design Representation Tool* sends a notice message with all information about this mechanical assembly and a predefined pattern, and also an ELM message with the same information to the agent *Graphical Tool2*. The agent *Graphical Tool1* receives this message and transforms all information into a DXF file for AutoCAD. The agent *Graphical Tool2* receives this ELM message and transforms all information into the format IGES for I-DEAS. If the process engineer makes a modification to this mechanical assembly because of a technological constraint or other technological restrictions, the *Process Engineer* will send an ELM message to the agent *Design Representation Tool* for asking a modification, which is still under development. Agent *Monitor* receives all above mentioned messages, displays them on a specific interface, and stores the information about the task being performed and the capabilities of the other agents in a temporal local knowledge base. Each agent has to register its capabilities when it connects to the design environment.

As a result from our experiments, it appears that $ToolTalk^{TM}$ is more intended to synchronize tools than to really serve as a communication medium for actors that want to talk with one another. In practice, it centralizes messages, thus defeating somehow the purpose of distributed control. However, on an open network of workstations, one must anyway limit the diffusion of broadcasted messages, meaning that there has to be somewhere a list of machines that are potential receivers. To implement a true broadcast, the mechanism would have to be changed.

5. Conclusion and Future Work

The objective of our research is to develop a distributed intelligent design environment for supporting cooperation among the existing engineering tools. The techniques of Distributed Artificial Intelligence provide a natural model for such applications. In this paper, we discussed a distributed architecture for intelligent design environment based upon asynchronous cognitive agents combining a number of mechanisms already found in the literature. Such an architecture is specially useful for large design problems. It also offers some important features such as modularity, flexibility, extensibility and transportability.

We have implemented an experimental distributed intelligent design environment by employing an object-oriented communication module called *ToolTalk*^{TM} and an ELM mail system to support the cooperation among the engineering tools organized as independent agents. In this environment, we have successfully implemented communications among seven independent agents located in the different workstations and microcomputers, which proves that the architecture proposed in this paper is feasible. However, as a result of our experiments, we found that commercially available products like *ToolTalk*^{TM}, although usable, are not the best support for this kind of approach, and that new tools are required. Our goal, as mentioned previously, is to obtain first hand experience in the case of large design projects.

DIDE is an ongoing research project. Future work in several areas is now in progress: (1) better use of the version mechanism of MOSS for conflict resolution in the agent *Project Manager*; (2) a distributed multimedia database for facilitate the information exchanging among the agents; (3) additional agents such as *Process Engineer* for verifying the technological constraints, *Technological Database* for a special manufacturer; *CAPP* for extending CAD to CAM, etc.; (4) development of a special communication protocol by combining the *ToolTalk*^{TM} and KQML (Finin, McKay, & Fritzson 1992) (project OSACA).

References

Abriat, P. 1991. Conception et réalisation d'un système multi-agent de robotique permettant de récupérer les erreurs dans les cellules flexibles. Thèse de Doctorat, Université de Technologie de Compiègne.

Barthès, J.P. 1989. MOSS: a multi-user object environment. In Proceeding of 2nd Symposium on AI, Monterrey, Mexico.

Barthès, J.P. 1993. La problématique de reconciliation en ingénierie simultanée. In Actes 01 DESIGN'93, Tunis.

Boissier, O. 1993. Problème du controle dans un système intégré de vision: Utilisation d'un système multi-agent. In Actes de la 1ère journées francophones Intelligence Artificielle Distribuée et Système Multi-Agents, Toulouse, France.

Cavalli, A.; Hardin, J.; Petrie, C.; Smith, R.; and Speyer, B. 1991. Technical Issues of Enterprise Integration, Research Report EID-349-91, MCC, Austin, Texas.

Cutkosky, M.R.; Engelmore, R.S.; Fikes, R.E.; Genesereth, M.R.; Gruber, T.R.; Mark, W.S.; Tenenbaum, J.M.; and Weber, J.C. 1993. PACT: An Experiment in Integrating Concurrent Engineering Systems. *IEEE Computer*, January 1993: 28–37.

Finin, T.; McKay, D.; and Fritzson, R. 1992. Specification of the KQML: Agent-Communication Language, Tech. Report EIT TR 92-04, Entreprise Integration Technologies, Palo Alto, Calif., USA.

Frankel, R. 1991. The ToolTalk Service, A Technical Report, SunSoft.

Gero, J.S. 1987. *Expert systems in computer-aided design*. Elsevier Science Publishers B.V., North-Holland, Amsterdam.

Gruber, T. 1993. A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition* 5(2): 199–220.

Iffenecker, C. 1994. Modélisation et réutilisation d'expertises variées en Conception. In Actes du Séminaire Interdisciplinaire "Mémoire Collective", Compiègne, France.

Lee, K.C.; Mansfield, W.H.Jr.; and Sheth, A.P. 1993. A Framework for Controlling Cooperative Agents. *IEEE Computer*, July 1993: 8–16.

Monceyron, E. and Barthès, J.P. 1992. Architecture for ICAD Systems: an Example from Harbor Design. *Revue Science et Techniques de la Conception* 1(1): 49–68.

Morse, D.V. 1990. Communication in Automated Interactive Engineering Design. PhD thesis, Carnegie Mellon University.

Park, H.; Cutkosky, M.R.; Conru, A.B.; and Lee, S.H. 1994. An Agent-Based Approach to Concurrent Cable Harness Design. *AIEDAM* 8(1).

Petrie, C.; Cutkosky, M.; and Park, H. 1994. Design Space Navigation as a Collaborative Aide. In Proceeding of the Third International Conference on AI in Design, Lausanne.

Populaire, Ph.; Demazeau, Y.; Boissier, O.; and Sichman, J. 1993. Description et implémentation de protocole de communication en univers multi-agents. In Actes de la 1ère journées francophones Intelligence Artificielle Distribuée et Système Multi-Agents, Toulouse, France.

Quadrel, R.W.; Woodbury, R.F.; Fenves, S.J.; and Talukdar, S.N. 1993. Controlling Asynchronous Team Design Environments by Simulated Annealing. *Research in Engineering Design* 5(2): 88–104.

Ribeiro Gouveia, F. and Barthès, J.P. 1993. Cooperative Agents in Engineering Environments. In Proceedings of EuropIA'93 Workshop on Intelligent Information Environments, Delft.

Scalabrin, E. and Barthès, J.P. 1993. OSACA, une architecture ouverte d'agents cognitifs indépendants. In Actes de la Journée "Systèmes Multi-Agents", Montpellier, France.

Shen, W.; Barthès, J.P.; and EL Dahshan K. 1994. Propagation de Contraintes dans les Systèmes de CAO en Mécanique. *Revue internationale de CFAO et d'infographie* 9(1-2): 25-40.

Sriram, D.; Stephanopoulos, G.; Logcher, R.; Gossard, D.; Groleau, N.; Serrano, D.; and Navinchandra, 1989. D. Knowledge-based Systems. Applications in Engineering Design: Research at MIT. *AI Magazine* 10(3): 79-96.

Sriram, D.; Logcher, R.; Wong, A.; and Ahmed, S. 1991. An object-oriented framework for Collaborative Engineering Design. In Lecture Notes in Computer Science, pp.51-92. Springer-Verlag.

Sriram, D.; Logcher, R.; Groleau, N.; and Chernoff, J. 1992. DICE: An Object Oriented Programming Environment for Cooperative Engineering Design. *AI in Engineering Design* Vo.3, Tong C. and Sriram D. (Eds.), Academic Press.

Stephanopoulos, G.; Johnston, J.; Kriticos, T.; Lakshmanan, R.; Mavrovouniotis, M.; and Siletti, C. 1987. DESIGN-KIT: An Object-Oriented Environment for Process Engineering. *Computer in Chemical Engineering* 11(6).

Thoraval, P.; and Mazas, Ph. 1990. ARCHIX, an Experiment with Intelligent Design in the Automobile Industry. In Proceeding of 4th Eurographics Workshop on Intelligent CAD Systems, Mortefontaine, France.

Toye, G.; Cutkosky, M.; Leifer, L.; Tenenbaum, J.; and Glicksman, J. 1993. SHARE: A Methodology and Environment for Collaborative Product Development. In Proceeding of 2nd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Press, pp.33-47.

Trousse, B. 1993. Towards a multi-agent approach for cooperative distributed design assistants. In Proceeding of EuropIA'93 Workshop on Intelligent Information Environments, Delft.

Van Dyke Parunak H. 1990. Toward a formal model of inter-agent control. In Proceeding of 10th AAAI Int'l Workshop on Distributed Artificial Intelligence, Bandera, TX.