AN EXPERIMENTAL 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

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 often for quite a long time with different tools located on various sites. In order to ensure the coordination of design activities in the different groups or the cooperation among the different tools, it is necessary to develop an efficient 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 an agent architecture and inter-agent communications are discussed. A prototype of such an environment with a number of independent agents located in several workstations is then presented and demonstrated on an example of a small mechanical design.

1. Introduction

Real world engineering design projects require the cooperation of multidisciplinary design teams using sophisticated and powerful engineering tools such as commercial CAD tools, engineering databases, knowledge-based systems, simulation systems and other special purpose computational tools. The individuals or the individual groups of the multidisciplinary design teams work in parallel and independently often for quite a long time with tools located on different sites. On the other hand, at any instant, individual members may be working on different versions of a design, viewing it from various perspectives (e.g., electronics, manufacturing, planning), at various levels of details. In order to coordinate the design activities of the groups and to guarantee a good cooperation among the engineering tools, it is necessary to develop efficient distributed (eventually 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 to coordinate their actions as they explore alternatives in search of a globally optimal or near-optimal solution.

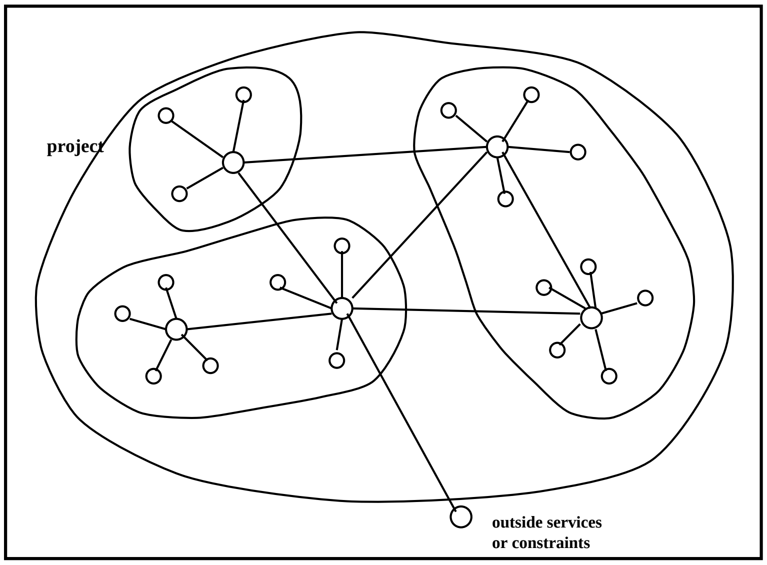

Fig 1. Desitributed design environment for large engineering projects

Our approach to large engineering projects is to decompose the overall task of designing a complex artifact into a set of different services. Such services are used by local teams which, in turn communicate with other teams. Typically such teams would reside at different locations and be specialized in different aspects in the design of the product. On Fig 1, the global shape corresponds to the design project, inside groups symbolize local sets of specialists which could include typically subcontractors. Local groups could work on a local network, and the groups could also share design data and knowledge via Internet. We do not plan to enforce a global consistency on the project. Each subgroup needs a number of services, which are usually closely dependent on engineering activities. Such services are to be assigned to agents both human and automatic. In this paper we only consider the part of the system which deals with the computerized agents. The overall project organization, global and local consistency are outside the scope of this paper. However, to fix ideas we would say that typically local exchanges will occur every second, and intergroup exchanges will occur every week.

Focusing on the work of a local set of specialists, we see the need for several types of services, which today are solved by engineering tools. At this level of our approach, as described in this paper, we find a number of other proposals aimed at using distributed problem solving techniques for addressing concurrent design.

Indeed, in the past ten years, a number of researchers have proposed to use distributed problem solving technology for concurrent design (Gero 1987; Morse 1990; Sriram et al 1991); or proposed some general architectures or platforms for engineering or industrial applications such as SHADE (McGuire et al 1993), SHARE (Toye et al 1993), ARCHON (Jennings, Corera, & Laresgoiti 1995; Cockburn &

Jennings 1995), SINE (Brown et al 1995), Synchronous Group Applications (Tou et al 1994), PACO (Demazeau 1993) and OSACA (Scalabrin & Barthès 1993); 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), ABCDE (Balasubramanian & Norrie 1995), SiFA systems (Brown et al 1995; Dunskus et al 1995), 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), IBDE (Fenves et al 1990), ANAXAGORE (Trousse 1993), CONDOR (Iffenecker 1994), EXPORT (Monceyron & Barthès 1992), ARCHIX (Thoraval & Mazas 1990), some specific computer tools for inter-agent communication such as KQML (Finin, McKay, & Fritzson 1992), COOL (Barbuceanu & Fox 1995), $ToolTalk^{TM}$ (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).

Here, we distinguish between two types of cooperative design systems according to their organization: multi-expert systems (based on a blackboard structure) and multi-agent systems. In a multi-expert system each specialist (sometimes called a knowledge source) has access to the blackboard when all information is posted and made available to all. In a multi-agent system each agent is independent and has its own representation of the situation independent from that of other agents. In the first case one normally uses a shared memory, and in the second case agents can be moved freely in independent machines provided that they can communicate using messages. It is more than a simple difference in implementation. A blackboard architecture has the definite advantage of ensuring consistency, it has nevertheless some drawbacks. Firstly, the blackboard can be a communication bottleneck. Secondly, when using a blackboard to coordinate different engineering tools, each tool has its own representation of the data describing the problem to solve; such a representation is often private to the tool, and there is no particular advantage in transferring it to a centralized location. Thirdly, during a project lifetime new tools may be introduced, changing the structure of the design environment. This architecture was used in some engineering design projects, 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). It was also used in the ITX system (Lee, Mansfield, & Sheth 1993) for a teleconferencing environment. An example of such an approach is given in Fig 2 showing the structure of EXPORT.

On the contrary, in Multi-Agent Systems, each agent builds its own model of the current solution by acquiring information from the other agents. This type of system needs a communication protocol and a message format (common language) according to which the requests and the responses will be formed. Each agent stores the current solution, or at least a part of this solution in its local fact base. Generally, in multi-agent systems, agents are heterogeneous and autonomous, i.e., there is no global control; the communications among agents can be synchronous or asynchronous, and can be point-to-point, broadcast, or multicast; and the semantics of the messages among the agents is high. Some projects like MARS (Abriat 1991), PACT (Cutkosky et al 1993), First-Link (Park et al 1994) and Next-Link (Petrie, Cutkosky, & Park 1994) have used this architecture.

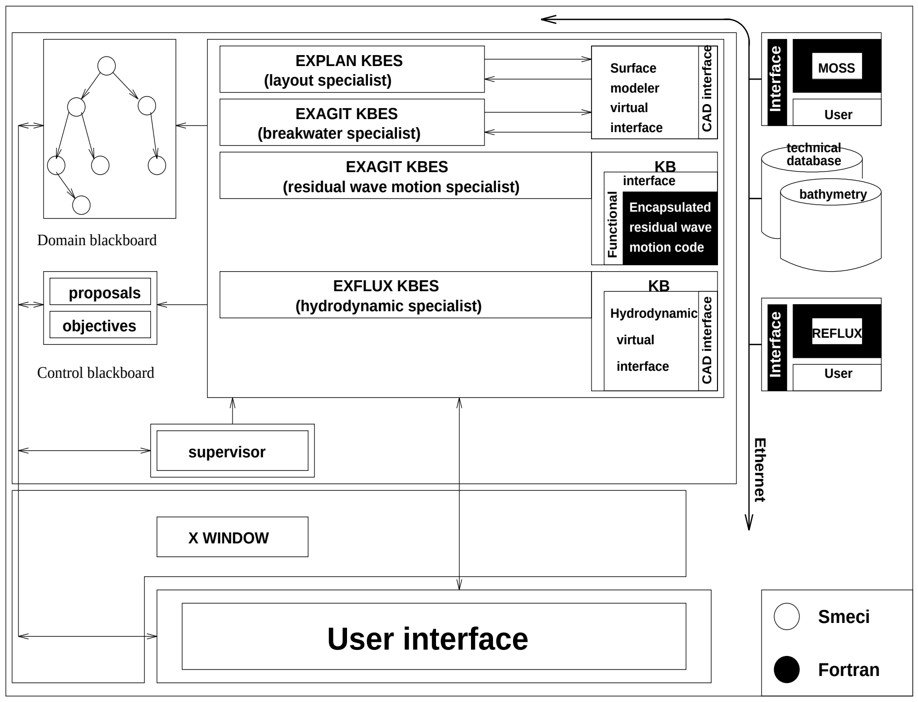
EXPLAN KBES (layout specialist)
EXAGIT KBES (breakwater specialist)
Surface modeler virtual interface
CAD interface
EXAGIT KBES (residual wave motion specialist)
KB interface
Functional
Encapsulated residual wave motion code
EXFLUX KBES (hydrodynamic specialist)
KB
Hydrodynamic virtual interface
CAD interface
Domain blackboard
proposals
objectives
Control blackboard
supervisor
X WINDOW
User interface
Interface
MOSS
User
technical database
bathymetry
Interface
REFLUX
User
Ethernet
Smeci
Fortran

Fig 2. A general architecture of the EXPORT system

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 fine in very large design projects of complex systems such as an automobile, a locomotive, a harbor, or an aircraft. Such design projects share the following 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, we do not think that a multi-expert architecture built upon a blackboard architecture could scale up. On the other hand, developing a network of independent local agents looks more promising. A major difference between our approach and that of SHADE-based projects (PACT, First-Link, Next-Link) is in the global structure. Indeed, in DIDE all agents are independent and first class; in particular, there is no facilitator structure like in PACT.

The rest of the paper describes our project. It 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 the experimental implementation of the DIDE with a small mechanical design example. Section 5 discusses some important issues. And finally section 6 gives some concluding remarks.

2. Agent Architecture and Inter-Agent Communication

2.1. Internal Structure of an Agent

In our 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 agent. Fig 3 shows the internal structure of an agent (Ribeiro Gouveia & Barthès 1993).

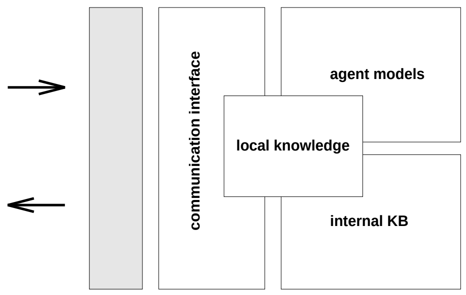

Fig 3. Internal structure of an agent

An agent is composed of: (i) a network interface (shaded portion in Fig 3); (ii) a communication interface; (iii) symbolic models of the other agents, and associated methods to use them (agent models in Fig 3); (iv) a model of its own expertise (internal KB); (v) a model of the task to be performed, or of the current context (local knowledge).

- The network interface simply couples the agent to the network.

- The communication interface of an agent is composed of several methods or functions for communicating with other agents by sending and receiving messages. A message box is used to store temporarily all received messages. Processing incoming mails requires two steps: (i) receiving, storing, and sorting messages; (ii) encoding a message content for further processing by the agent in the context of a particular task. Processing ongoing mail similarly requires encoding of the information to be transmitted, and actually mailing it according to the exchange protocol.

- The symbolic models of the other agents are realized by means of a knowledge base containing information about the other agents, obtained during interaction. The information includes an agent's name, address, and skills or competences. The knowledge helps the current agent to select one or more agents as sub-contractors for processing actions.

- The model of "own expertise" is also a knowledge base composed of self information such as the name, address and competences or skills. The latter may be methods for activating actions corresponding to the received requests.

- The local knowledge is a knowledge base composed of the information about the task to be performed. It contains some already achieved partial solutions for a particular task.

At first, when a new 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. In the case of slave agents (e.g., a database containing a list of part prices) this will not change, i.e., it will remain empty. Every other agent must build 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.

2.2. Inter-Agent Communication

In general, communication can be synchronous or asynchronous, and the communication mode can be point-to-point (between two agents), broadcast (one to all agents), or multicast (to a selected group of agents). The OSACA-based DIDE environment uses a multi-cast mode which can be extended to point-to-point and broadcast. In addition, DIDE allows for five simple actions: request, inform, announce, bid and notice. They are grouped into two categories: requests and assertions (REQUEST, INFORM, NOTICE); call for bids and propositions (ANNOUNCE, BID).

3. A General Architecture for Design Environment

The overall organization of an agent-based architecture for design was described in the introduction. Several issues need to be clarified.

The system is not intended to run automatically. On the contrary, human beings are part of the system. The system cannot be organized independently of the company structure. Thus, we assume the project will be run by a project manager, and that each local group will in turn have a local project manager. We are interested here in a local group. Within the local group, some agents will be controlled by humans, other will provide services automatically. When humans are in full control, then this amounts to a traditional client/server architecture. It becomes more interesting when some agents are actively offering services when they know the final goal, i.e., the agents not only reply to the requests of the other agents, but also can give some suggestions or propositions to the current design project according to their knowledge.

A second issue relates to the control of the design task. Apart from the project manager we assume no central control of the design task. Subtasks are created for answering requests as needed, once the work is initiated. There is no global planning either central or distributed. From this point of view, the agents are purely reactive. However, some subtasks may have predefined sequences (scenarios, amounting to local plans). When this is the case, there is no reason why such sequences could not be used.

A third issue concerns the global consistency of the design. The consistency is kept at a minimum, considering that each agent has a local model of the designed device. However, in mechanical design it is sometimes possible to show a model of the designed product. In such a case, each subgroup is allowed to modify the model independently, using its own design space. Results are kept in different versions of a model database. Then, at each progress meeting, it is the responsibility of the product manager to merge all the version propositions into a unique design. Of course, the environment has to support the merging process (reconciliation) efficiently (Barthès 1993).

A fourth issue concerns the agent behavior. 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. Thereafter, it receives messages like any other agents.

A fifth issue is related to the legacy problem. 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 (efficient only when 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

A small system has been developed for testing the basic feasibility of the project, as well as the possibility to encapsulate traditional tools, and to bring agents in and out without halting the system.

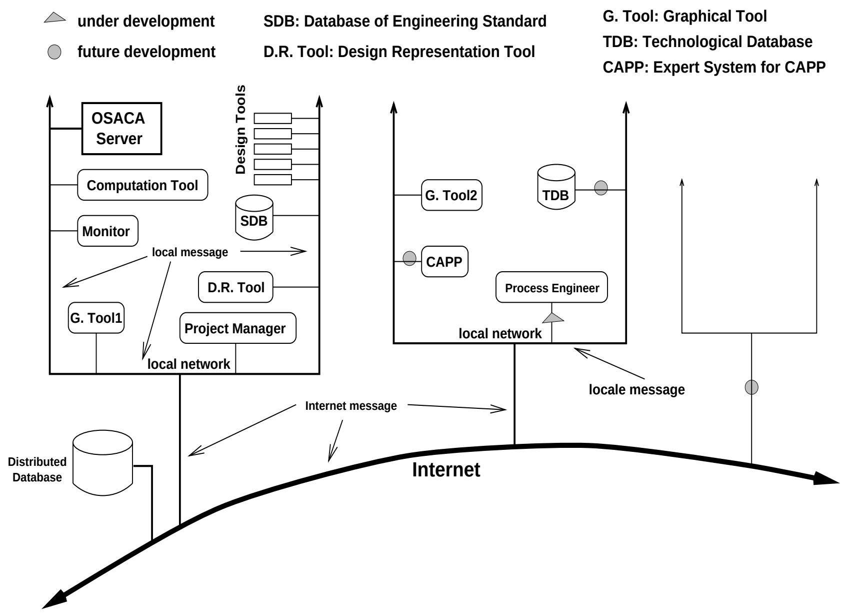

Fig 4. An Experimental System Architecture of DIDE

The first version used only slave agent (no active agents) and looked like a generalized client/server architecture. The current version, still being developed, uses the OSACA platform. Although developed as a local group, i.e., using several workstations on a local network, it has the capability of intergroup communication using Internet.

4.1. Hardware and Software Environment

The experimental environment is built on a network of SUN and VAX workstations. The OSACA platform developed in the DAI research group of University of Technology of Compiègne (Scalabrin & Barthès 1993) is used for defining agents and inter-agents communications. 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. Two databases $MATISSE^{TM}$ (an object-oriented database, commercial product of ADB) and EDBMS (an extended relational database developed in Chinese Academy of Sciences) are used for storing design data and knowledge.

4.2. Description of DIDE Agents

The current version of DIDE contains a number of agents, including Project manager, Monitor, Database of Engineering Standard, Object-Oriented Distributed

Database, Graphical Tools and a group of Design Tools (as shown on Fig 4):

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 conflicts 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 from different design groups or from different domains such as dynamics, electronics, economics and process engineering for discussion and negotiation.

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. All received messages are displayed on a text interface and a graphical interface is used to show the current system situation. If a new agent is connected to the system, an icon appears on the graphical interface, and if an old agent is removed from the system, the corresponding icon disappears. If there are some messages passing between agents, some signs will be showed on the graphical interface for a short time.

Design Representation Tool: This agent is responsible for the generation and management of geometric entities of a particular mechanical system. All classes, instances and methods are defined in this tool by using an object-oriented language, MOSS. The objects can be stored directly in a local extended relational database EDBMS or a distributed object-oriented database $MATISSE^{TM}$.

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

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.

Following five *Design* agents are developed specially for an experimental design example - Assembly Design. The agent *Assembly Design* is used for global design, assembling the parts, propagating constraints and verifications. Four other agents are used to carry out sub-tasks (parts design) in parallel. For large design projects, the agent *Assembly Design* is corresponding the global system design, and the other agents should be developed to accomplish the design of the sub-systems, or sub-sub-systems.

Assembly Design: This agent is used to take the global design of the assembly. It decomposes the global design task into some sub-tasks and asks some special agents to accomplish the sub-tasks. The constraints propagation and verification are performed by this agent. Therefore, it is responsible for detecting the conflicts by comparing the results obtained from the agents carrying out the sub-tasks. If some simple conflicts are detected, they may be resolved by some criteria in this agent. If not, a request message is sent to *Project Manager* for asking for a decision. At the end of a design or modification, the results can be showed on a 2D graphical interface, which is developed on LispView, and which will be replaced by an interface developed on CLIM.

Bolt Selection: It is used to take the sub-task sent by *Assembly Design* for completing bolt selections. It shares mechanical objects with other concerned agents through a local database EDBMS or through *Distributed OODB MATISSE*. The current function of this agent is to standardize a bolt according to a calculated bolt diameter provided by *Project Manager*.

Nut Selection and *Washer Selection*: For the sake of the example, these two agents are used to take the sub-tasks sent by *Assembly Design* for completing nut and washer selections, which are similar to the agent *Bolt Selection*.

Plate Design: This agent is used to take the sub-task sent by *Assembly Design*. It receives the REQUEST messages from *Project Manager* to complete plate designs, e.g., calculating hole diameters and deciding the thicknesses of the plates. It also shares objects with other concerned agents through databases (local or distributed).

Graphical Tool1: Currently, $AutoCAD^{TM}$ on a SUN^{TM} workstation with SOLARIS 2.3 is used to display the design results on a 3D graphical interface. It can be also used to prepare a mechanical drawing or to let the drawings be modified by a human domain specialist. In this case, the agent is taken as a human agent and the main window of $AutoCAD^{TM}$ can be taken as a user interface for human specialists after some development on AutoLisp (This is not the case of current implementation, but is a future development). The data exchange with other agents is done by means of standard DXF (Drawing Interchange Format) and IGES files.

Graphical Tool2: This graphical tool is $SDRC^{TM}$ $I-DEAS^{TM}$ located on the network of DEC^{TM} workstations. This agent communicates with other agents by means of an electronic mail system. This agent is used to test communications among agents located on different local networks and with different operating systems. The experimental goal is also to create graphical interfaces for process engineers to verify technological constraints.

Distributed OODB MATISSE: $MATISSE^{TM}$ is a commercial object-oriented database of ADB. An agent has been created by using the kernel system of $MATISSE^{TM}$ in the OSACA project for storing MOSS objects created and used by all OSACA-based agents. In DIDE, we use it as a second potential database for storing the mechanical objects shared by concerned agents. Another available database is EDBMS as mentioned in the paragraph about *Design Representation Tool*.

4.3. Inter-Agent Messages

In the current DIDE implementation, only two types of messages (REQUEST and INFORM) are used for inter-agent communications. REQUEST messages can be sent by a method `=request`, and responses can be obtained by a method `=get-result`, e.g.,

```
(setq msg1-sent (-> *COM* '=request :receiver "agent3"
                :ontology "get-standard-bolt-dimension"
                :content (list 'GETSTB tabname d)))
(setq ans1 (-> *COM* '=get-result msg1-sent))
```

INFORM message can be sent in the same fashion by a method `=reply`. Before being sent out, REQUEST or INFORM messages are filled by some default values such as `:sender, :id, :language` by the current agent.

Here are some examples of the two types of messages.

agent2:

```
Msg-sent : AGENT3 REQUEST :id 816564885734132 :priority 1 :sender agent2
:receiver AGENT3 :language MOSS :ontology get-standard-bolt-dimension
:commode 0 :reply-with OREQ1 :content (GETSTB TABB.112 17)
Msg-received : INFORM :id 816564888728152 :priority 1 :sender agent3
:receiver AGENT2 :language MOSS :ontology get-standard-bolt-dimension
:commode 0 :in-reply-to OREQ1 :content (8 20 30 13)
```

agent3:

```
Msg-received : REQUEST :id 816563689462963 :priority 1 :sender agent2
:receiver AGENT3 :language MOSS :ontology get-standard-bolt-dimension
:commode 0 :reply-with OREQ1 :content (GETSTB TABB.112 17)
Msg-sent : AGENT2 INFORM :id 816563692192752 :priority 1 :sender agent3
:receiver AGENT2 :language MOSS :ontology get-standard-bolt-dimension
:commode 0 :in-reply-with OREQ1 :content (8 20 30 13)
```

4.4. Agent Models

As mentioned in Section 2, agent models including the symbolic models of other agents and the model of its own expertise are very important for cognitive agents. In the experimental DIDE environment, the "model of own expertise" for an agent is a simple knowledge base composed of the information about itself such as its name, address and competences or skills which may be some methods for activating some actions corresponding to the received requests. This simple knowledge base is stored as a MOSS object, instance of the class KNOWLEDGE:

```
Entity-name            KNOWLEDGE
Terminal-Property      HAS-AGENT-NAME
                       HAS-ADDRESS
                       HAS-ACTIVE?
Structural-Property    HAS-SKILLS SKILL
```

where the class SKILL:

```
Entity-name            SKILL
Terminal-Property      HAS-SKILL-NAME
                       HAS-KEY-WORDS
                       HAS-EXPLANATIONS
```

As a simple example of the above mentioned agent *Computational Tool*:

```
HAS-AGENT-NAME      agent4
HAS-ADDRESS         kaa.hds.univ-compiegne.fr
HAS-ACTIVE?         yes
HAS-SKILLS          skill.1
```

where skill.1

```
HAS-SKILL-NAME      CALCULATE
HAS-KEY-WORDS       bolt-diameter-calculation
HAS-EXPLANATIONS    "obtaining a bolt diameter through
                     bolt stress calculation"
```

The symbolic models of other agents are implemented by means of a knowledge base containing information about other agents obtained during interaction with the other agents. The information includes the agent's name, address, and its skills or competences. The same object structure is used to store this information about other agents. The knowledge helps the current agent to select one or more agents as sub-contractors for processing actions. The current message passing mechanism for obtaining this information is as follows: as soon as a new agent connects to the system, it broadcasts a message of REQUEST type and with context "REGISTER" to all other agents. Each one of the other agents receives this message, extracts the information of interest concerning the new agent such as its name, address and skills, and uses it to update its agent models, and then sends an "INFORM" message to the new agent. The new agent receives "INFORM" messages from all other agents, extracts important information and uses it to update its agent models.

4.5. A Scenario - A Small Example

As a design example, we considered a mechanical assembly, already described in details in (Shen, Barthès, and EL Dahshan 1994). Supposing that the necessary DIDE agents have all been connected to the system and each agent has established its agent models by message passing as mentioned in Section 4.4.

In the current implementation, only two types of messages (REQUEST and INFORM) are used. This paragraph shows how an agent handles such messages (see Fig 5). When an agent receives a REQUEST message for some sub-task, this agent chooses a method according its internal KB (knowledge about itself), executes this method for completing the sub-task, and then sends the results by INFORM message to the sender (agent) of the REQUEST message. During the execution of the method, this agent may have to ask some other agents to complete part of the sub-task (sub-sub-task). It chooses an agent B according to its agent models (knowledge about other agents) and sends a REQUEST message to the agent B. After receiving the results by INFORM message sent by agent B, the agent resumes its work. During the work, some important information is saved in its local knowledge base for future use. A more complicated mechanism has been used to select an agent for carrying out a sub-task or sub-sub-task, which was described in details in (Shen and Barthès 1995).

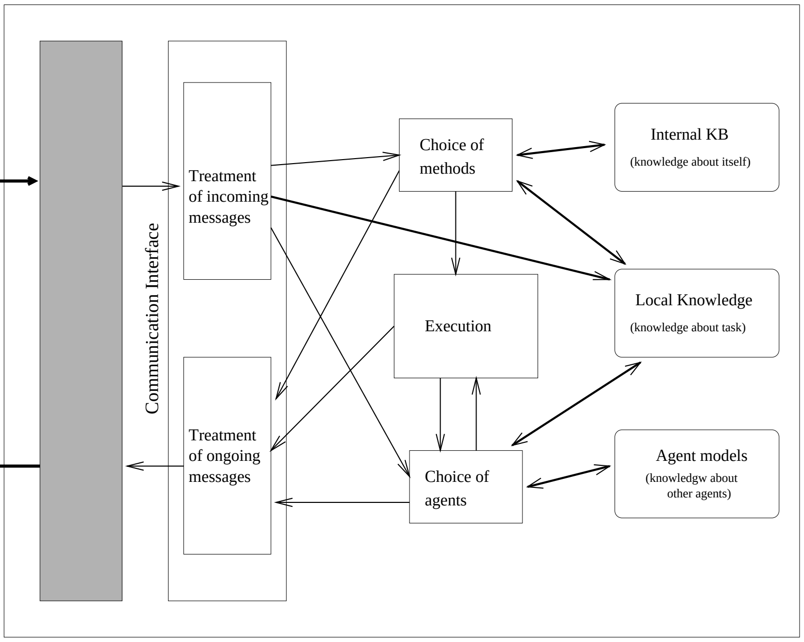

Fig 5. Treatment of messages by an agent

At the beginning of the design or re-design process, the agent *Project Manager* sends a REQUEST message to the agent *Design Representation* to define all necessary objects including classes, instances and methods and store them in the local database EDBMS or distributed database. If such objects exist in the local database or distributed database, *Design Representation* sends simply an INFORM message to *Project Manager* with `content` *OK*. If not, *Design Representation* defines all objects and stores them in the databases, and then sends an INFORM message to *Project Manager*.

If the project manager desires to design or re-design the 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 REQUEST message with initial data to *Computational Tool* according its knowledge that *Computational Tool* can carry out this calculation and then waits for a response. If it does not receive a response for a period which can be known by the knowledge about the agent *Computational Tool* and associated SKILLS, it repeats the REQUEST twice. And if there is still no response, the agent *Project Manager* considers that *Computational Tool* is not active and the design task cannot be done at this time. In the current implementation of DIDE, most of agents work on this type of mechanism. In the next paragraphs, we suppose that all concerned agents are active and that each agent has decided which agent must be contacted and which SKILLS must be used according its agent models.

The agent *Computational Tool* receives the REQUEST message and does the calculation. Upon completion of the calculation, *Computational Tool* sends an INFORM message with the result to *Project Manager*. Here the result is a calculated bolt diameter.

After receiving this message, the agent *Project Manager* then sends a REQUEST message to *Assembly Design* for designing or modifying the assembly. *Assembly Design* decomposes this design task into sub-tasks and sends multi-cast type REQUEST messages in parallel to four design or selection agents *Bolt Selection, Nut Selection, Washer Selection* and *Plate Design*. These four agents carry out the sub-tasks (selection or calculation) in parallel according to the calculated bolt diameter. During the selection or calculation process, each of these agents first loads the initial object data from local or distributed database, and then modifies them according the standardized or calculated results, and finally stores them into local database and distributed database if all these databases are active. In the current experimental example, each of these agents tries to obtain data from the local database first if they exist, loads them; but if not, it tries to obtain them from the distributed database. The results are stored into two databases whenever they are active. After completing the sub-tasks, each of these four agents sends an INFORM message to *Assembly Design*. The `content` of the INFORM message from three *selection* agents contains only the standardized diameter, and that from *Plate Design* is *OK*. During the selection process the three *Selection* agents need standard tables to obtain standard (nominal) diameter and other standard dimensions of the bolt or the nut or the washer. They send a REQUEST message with a calculated diameter to *Database of Engineering Standards* and wait for a reply. The agent *Database of Engineering Standards* receives this type of messages and searches for the requested information in the database of national engineering standards and then sends an INFORM message with the results to requesting agents.

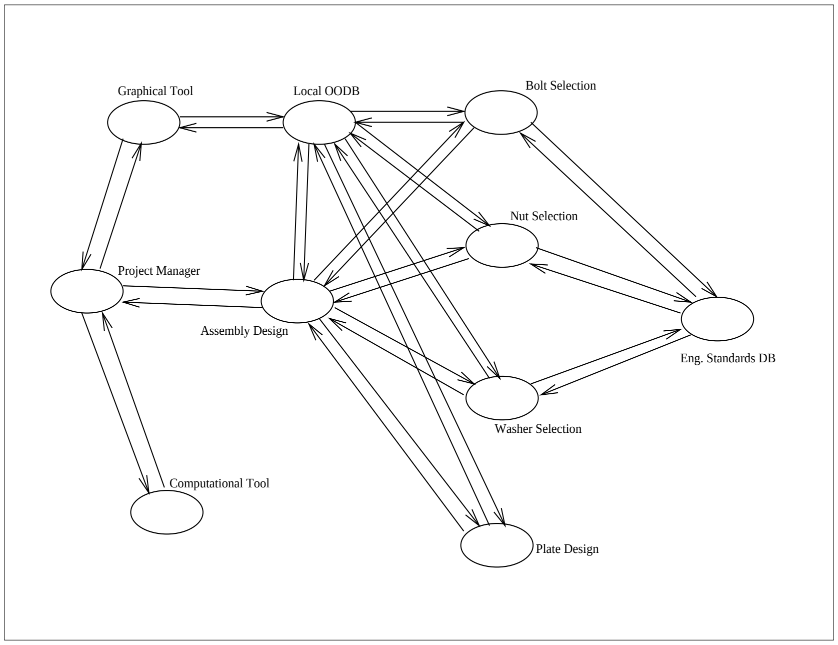

Fig 6. Possible message paths among the autonomous agents of DIDE

The agent *Assembly Design* receives the messages from four agents, analyses and compares the results. If the standardized diameter of the bolt, that of the nut and that of the washer are not consistent, *Assembly Design* has to make a decision to select one diameter as the "calculated bolt diameter". It sends a multi-cast type REQUEST messages to the design and selection agents again. Here the decision is taken by selecting the largest of the three diameters in this special mechanical example. If there is no conflict among the received results, *Assembly Design* verifies the global constraints and may modify some object data such as the bolt length and the bolt thread length. The object data have to be loaded from the local or distributed database before the verification, and stored into these database after that verification. And finally, *Assembly Design* displays the results on a 2D graphical interface and sends an INFORM message to *Project manager* with `content` *OK*.

The *Project manager* receives the INFORM message from *Assembly Design* and may send a message to *Graphical Tool1* for displaying the results which can be obtained from the databases on a 3D graphical interface, and may also send an E-mail with the results to *Graphical Tool2*. The agent *Graphical Tool2* receives this message and transforms all information into the format IGES for I-DEAS, which will be used for process engineers in future computation.

The *Monitor* receives all above messages sent by all other active agents in the same system. All received messages are displayed on a text interface. A graphical interface is used to show the current system status. If a new agent is connected to the system, an icon will be drawn on the graphical interface, and if an old agent is removed from the system, the corresponding icon will disappear. If there are some message passing between agents, some signs will be shown on the graphical interface for a short time.

Fig 6 shows examples of possible message paths among the main autonomous agents of DIDE.

5. Discussion

The experimental research of the project DIDE produced some useful insights that are applicable in future developments and in other research projects. Here is a brief discussion of some interesting issues.

5.1. Agent Life Cycle

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 months or even 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 approaches are 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.

In the OSACA-based DIDE system agents may appear or disappear with minimal disturbance to the whole system.

New agents, whether slaves (e.g., some encapsulated existing tools) or more active, are created from generic agents. They possess communication capabilities and some expertise (which may result from programs, rules, databases, etc.). They are not required to know anything about the existing agents, nor of the ongoing tasks to be solved.

When a new agent connects to the group, it immediately can receive broadcasted messages and respond to them. Other agents are not modified by the introduction of a new agent, i.e., the system keeps running.

Once a new agent is connected, it must build its own representation of the task being solved as well as acquire information about the other agents. Currently, two strategies can be used: (i) either a "tutor" agent helps the new agent to do it, - this amounted to downloading the information in the MARS project (Abriat 1991); or (ii) models will be build as needed through exchanged messages. Two things could be done in addition: (i) an agent could advertise its skills as soon as it is connected; (ii) an agent could inquire actively once it is connected by broadcasting requests. Such behaviors which can be installed in the generic agent from which more specific agents are derived, are not currently implemented in the OSACA agents.

An active agent may die (temporarily or definitively) or be removed from the system. In such a case, generally nothing special is done. In the current DIDE experimental environment, if an agent is normally disconnected, all other agents will note it and modify their agent models accordingly.

5.2. Cognitive Agent

An important objective of our research is to design cognitive agents for the DIDE project. 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; (iii) it is not simply a reactive one, i.e., it does not only reply to the requests from other agents, but it can also give some suggestions or propositions to the current solution according to its knowledge about the the current design task. This is a part of our ongoing research work.

5.3. Openness and Dynamics

For large engineering design projects, the design process often lasts for a long time, thus, the openness of a system is very important and truly central to the approach. 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. The OSACA-based DIDE environment implements this idea.

Here we give a scenario which we encountered during experimental stages for showing the openness and the dynamics of our system. During the assembly design process, after receiving design results from four sub-tasks-taking agents, the agent *Assembly Design* starts to verify the global constraints for assembly. During this verification, *Assembly Design* has to send a REQUEST message to *Database of Engineering Standards* for obtaining standard bolt length and bolt thread length. *Database of Engineering Standards* received the messages and called the corresponding functions (skills) to search standard data. But there was a problem in the function GETSTTL (for searching standard bolt thread length) and this agent was out of order and became non-active. We had to "repair" this agent by redefining this function and re-start the agent. Meanwhile, the agent continued to receive REQUEST messages, dealt with the received messages and sent out the results without disturbing the whole working environment. *Assembly Design* waited for results from *Database of Engineering Standards* for several minutes without response and subsequently repeated the REQUEST message twice. This mechanism allowed *Database of Engineering Standards* to receive the REQUEST message from *Assembly Design* and to complete the design process.

5.4. Synchronous and Asynchronous

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), we found that they are not necessary for the type of the design work we are considering because we would like to develop environments for large design problems, for which the design period is often very long and synchronous communications in this case are neither required, nor efficient or economical.

5.5. Shared Ontologies

An active agent system works by exchanging messages at a high semantic level. 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. Then, whenever a new agent enters the system, it is its responsibility to make sure that the system contains all the needed ontologies. If not, it must bring the missing ontologies.

5.6. 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 Section 4.2.2).

6. Conclusion

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 an intelligent design environment based upon asynchronous 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 a platform, OSACA, developed in our research group (Scalabrin and Barthès 1993) to support the cooperation among the engineering tools organized as independent agents. In this environment, we have successfully implemented communications among about ten independent agents located in the different workstations, which proves that the architecture proposed in this paper is feasible. However, as a result of our experiments in the first version of our implementation, we found that some commercially available products, although usable, are not the best support for this kind of approach. Our goal, as mentioned previously, is to obtain first hand experience in the case of large design projects.

DIDE is an ongoing research project. Some important issues are being studied and developed: remote communication, conflicts detection and resolution, cognitive agents and shared ontologies.

7. Acknowledgement

The authors would like to thank Edson Scalabrin who developed the OSACA platform and for his help during the implementation of the experimental system. The first author acknowledges his financial support from the Chinese Government Scholarship.

8. 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.

Balasubramanian, S. and Norrie, D.H. 1995. A Multi-Agent Intelligent Design System Integrating Manufacturing and Shop-Floor Control. In Proceedings of First International Confernece on Multi-Agent Systems, San Francisco, USA, The AAAI press / The MIT Press.

Barbuceanu, M. and Fox, M. 1995. COOL: A Language for Describing Coordination in Multi Agent Systems. In Proceedings of First International Confernece on Multi-Agent Systems, San Francisco, USA, The AAAI press / The MIT Press.

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.

Brown, D.C.; Dunskus, B.; Grecu, D.L. and Berker, I. 1995. SINE: Support For Single Function Agents, in AIENG'95, Applications of AI in Engineering, Udine, Italy.

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.

Cockburn, D. and Jennings, N.R. 1995. ARCHON: A DAI System for Industrial Applications. In Foundations of DAI, eds. G.M.P.O'Hare and Jennings, N.R., Wiley Interscience.

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.

Demazeau, Y. 1993. La plate-forme PACO et ses applications. In Actes de la 2ème Journée Nationale du PRC-IA sur les Systèmes Multi-Agents, Montpellier, France.

Dunskus, B.; Grecu, D.L.; Brown, D.C. and Berker, I. 1995. Using Single Function Agents to investigate Negotiation, in AIEDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Special Issue: Conflict Management in Design, Smith, I., Ed., Cambridge U.P.

Fenves, S.J.; Flemming, U.; Hendrickson, C.; Maher M.L.; and Schimitt, G. 1990. An Integrated Software Environment for Building Design and Construction, *Computer-Aided Design*, 22(1):27-36.

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.

Jennings, N.R.; Corera, J.M. and Laresgoiti, I. 1995. Developing Industrial Multi-Agent Systems. In Proceedings of First International Confernece on Multi-Agent Systems, San Francisco, USA, The AAAI press / The MIT Press.

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

McGuire, J.M.; Kuokka, D.R.; Weber, J.C.; Tenenbaum, J.M.; Gruber, T.R. and Olsen, G.R. 1993. SHADE: Technology for Knowledge-Based Collaborative Engineering, *Journal of Concurrent Engineering: Application and Research* 1(3).

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 2ème Journée Nationale du PRC-IA sur les 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.

Shen, W.; Barthès, J.P. 1995. DIDE: A Multi-Agent Environment for Engineering Design. In Proceeding of The First International Conference on Multi-Agent Systems, San Francisco, USA, The AAAI press / The MIT Press.

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 Cherneff, 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.

Tou, I.; Berson, S.; Estrin, G.; Eterovic, Y.; and Wu, E. 1994. Prototyping Synchronous Group Applications. *IEEE Computer*, May 1994: 48-56.

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.