Distributed Architecture to Support Dynamic Decision Processes

1. Introduction

We have developed a DSS prototype to support all decision points of an industrial organization. We have designed the DSS model base [1] by using a modular structure, where each module is thought out to bring support to a specific decision making task.

The decision points of an industrial organization are generally located on different geographic sites, each of them uses specific techniques and models, and usually, they only share some information derived from punctual decisions. Therefore, we infer that the DSS must be designed following the model base architecture, as a set of sub-systems that operate with a weakness coupling level. This description makes apparent an inherently distributed application that must be reflected by the DSS architecture. In this sense, the aim of our work has been to obtain an appropriate DSS architecture that should satisfy two design principles: the distributed nature of the application and weakness coupling level between sub-systems. In this way it is possible to obtain a flexible system that may be adapted to the organization way of working.

The architecture of a distributed system is a high level structure that presents the system in terms of components called domains. This structure describes the relationship among components. That is, how they cooperate with each other performing the global system behaviour. The requirement of each domain is specified through the inputs required by each domain from other domains to perform its cooperative action in the global operation of the DSS. The inputs of each domain that are derived by analysing the decision processes to be supported by it. Then, the end product of a distributed architecture design is a model that represents the interaction among domain. This interaction model can be defined by means of scenario analysis of the decision processes to be supported by the distributed system. As a result of this analysis, we infer that there are two types of decision scenarios that we will call predictable scenario and no predictable scenario.

We define as predictable scenario one that can be previously specified to design the interaction model of each module that integrates the DSS. In this way, the requirements of the decision process associated to this scenario will be included into the interface of each module.

We define as no predictable scenario one that is associated to dynamic decision processes that can not be foreseen at the design time.

In this work we present a DSS distributed architecture, which presents two mechanisms to support the interaction among geographically distributed components. Firstly, we present a brief description of one of the integration mechanisms based on the federation concepts. Secondly, we present the other interaction mechanism designed to support dynamic decision process requirements, which is based on the mobile agent technology. Finally, we describe the main functionalities of the mobile agents that integrate the DSS distributed architecture and some implementation details.

2. The DSS Distributed Architecture

In the introduction section above, we defined two types of decision processes to be considered at the time of designing a DSS distributed architecture, these are: static decision processes that define predictable scenarios, and dynamic decision processes that define non predictable scenarios.

In searching an architecture that reflects the decision processes to be supported by our DSS, we find it suitable to develop it with two different mechanisms to support the interaction among the DSS geographically distributed components. One mechanism designed to support the static decision processes is based on the federation concepts [2]. This mechanism proposes a model of interaction among domains based on the requirements specified from predictable scenarios. This is an event driven mechanism where each domain must specify both its responsibility for publishing the information it generates and its subscription to the information produced by other domains. In this way, by means of this mechanism that we call contracts-based mechanism, the architecture only establishes information dependence among system components that implicates both weakness coupling and domain autonomy conservation.

The interaction model of this distributed architecture mechanism is flexible in the sense that at the design time it can be adapted to particular requirements of decision processes of a particular industrial organization. But, once it is implemented, it defines a communication interface adapted to the requirements of these predictable scenarios. In other words, the interaction model implemented by this distributed architecture mechanism does not admit the modules to ask for and to receive new information types. In this way, the modules can not support any particular requirements that have not been foreseen at the design time.

To support any information requirements not foresee at the design time, we have proposed to include a second mechanism in the distributed architecture that is designed to support the dynamic decision processes based on a model that uses the mobile agent technology to support the interaction among domains. We will describe both mechanisms in following sections.

3. Contracts-based Mechanism To Support Static Decision Process

The components specified in this interaction mechanism of the distributed architecture are domains, gatekeepers and the highway. In the following paragraph, we provide a brief definition of each of them.

Domain: A domain may consist of an application or a set of strongly integrated applications that bring support to a specific area or activity of the organization. For example, planning, scheduling, supervisory control, etc. These applications are adapted to area functionality and its work way. Domains are frequently geographically distributed and they have a high level of autonomy. The latter implies that each domain can follow its own development policy incorporating new information technologies and new applications; all pointed to an increase in its functionality due to inner changes of the area, without affecting other domains. A domain does not know other domains' operations. A domain provides information required by other domains and consumes information received from other domains.

Responsibilities and duties of a domain that integrates the DSS are specified by contract. A contract specifies the data that the domain must publish and their format as well as the data that interests the domain and their format. Furthermore, a contract specifies the rules to categorise these data. These

contract elements are incorporated by means of several software components: domain interfaces, gatekeeper interfaces and message queues [3].

The domains that integrate the DSS only have information dependences. The information dependence occurs when the domain must send some information to other domains, as a result of an inner event. The information dependence results in both less coupling and greater level of autonomy among domains. However, process, transaction and information dependences among the applications can exist together within the same domain. The domains that integrate the DSS architecture are shown in figure 1. They define decision points to which the DSS brings support. We will not describe the DSS domains because they are not the scope of this work. The criteria we used for partitioning the global DSS are described in [4].

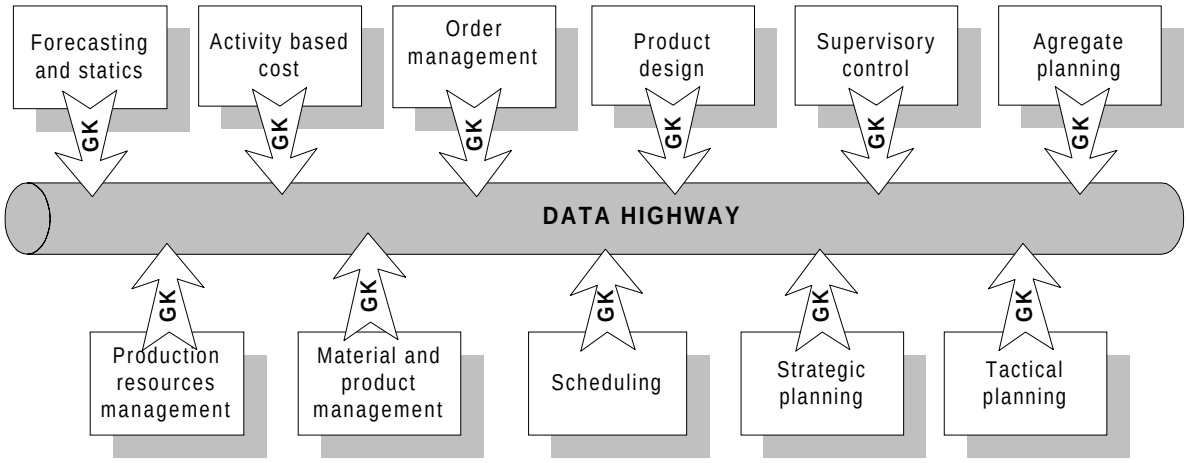

Fig. 1. DSS Distributed Architecture

Gatekeeper: Each domain has assigned a gatekeeper. It is a software component that interacts with the assigned domain and with other DSS gatekeepers. When a domain wants to provide information as result of an event, its gatekeeper will be responsible for sending this information. Then, the other gatekeepers will capture this information if it interests their respective domains. In this way, we can define the gatekeeper as a software component responsible for executing the actions of publishing information to other domains and consuming the information generated in other domains. Consequently, the DSS domains do not interact directly among them but by means of their gatekeepers. In this way, domains are freed to know and to work with the selected communication middleware mechanism. Gatekeepers are the ones that will use the middleware to execute the net information communication task.

Highway: It is the DSS architecture component that frees us from component communication details along the net. The highway will be constituted by an appropriated middleware to carry out the communication of information by the net. The middleware has to provide a set of services (such as: naming, data marshalling, event services, asynchronous communication service) that provides a high abstraction level. One of the objectives of this work has been to design a middleware independent architecture. For this reason, the gatekeeper plays an important role.

The interaction model of each domain is defined through its interface, which essentially summarises its behaviour in respect of other domains. Our aim has been not to adjust the design to a particular

communication technology. Thus, we specified the interface of each domain by means of primitive actions of sending and receiving information.

The contracts-based communication model of the DSS distributed architecture is based on an event oriented computation paradigm (Opposite to computation by requirement model on which the Client/Server systems are based [5]). This communication model allows the domain that produces the information to start interaction among domains. Also this model makes it possible to use the publishing/subscription paradigm. That is, in the presence of an event, domains will publish the information they produce for being consumed by other domains that are subscribed to this information.

By means of scenario analysis, we identified several interaction schemes among domains that integrate the DSS architecture. These interaction schemes describe the decision processes to which the DSS brings support [4]. The remittance of information between two domains originates an interaction that fixes the role of each domain. That is, a domain starts and finishes the interaction as transmitter, and the other domain starts and finishes the interaction as receptor. We present the design of the contracts-based mechanism components presented in [3]. This is out of the scope of this work.

4. Mobile Agents-Based Mechanism to Support Dynamic Decision Processes

The designed architecture is based on the use of mobile agents [6] to carry out the search and gathering of information [7][8]. Within the architecture itself, we can find a series of fundamental elements, namely: Collecting Agents, the Representative Agent, the Collecting-Agents Server, the Gatekeepers and the router. Now, we will provide basic notions about each of them. We present this architecture in figure 2.

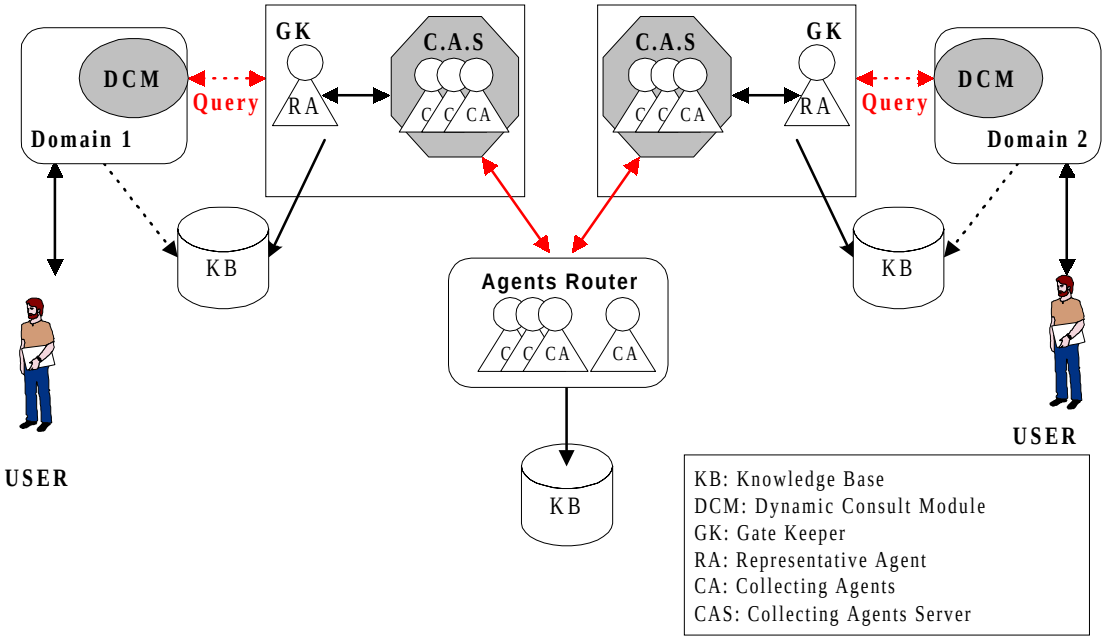

Fig. 2. Mobile Agents to support dynamic decision process

Collecting Agents (CA): The function of CA is to receive the user's consult and to obtain a list of domains that can provide an answer, according to the consult. Then, they must visit each of those domains gathering the available information, and once the collection phase is finished, they must return to the domain of origin to give the user the obtained information. The agents' behaviour during the collection phase is directly related to the privileges of the user making the consult, as it will be shown later on.

Representative Agent (RA): All domains have a RA that represents them. This agent is static and is located in the Gatekeeper of the domain, which must be always up and running. Its functions are the following: on the one hand, it sends the consults coming from users of the domain to the collecting agents so that the latter can search for the information. On the other hand, it transmits the answers to previous consults of the coming agents, and last, it represents the domain when a collecting agent that belongs to another domain arrives asking for information. To sum up, the representative agent of a domain acts as a mediator between a domain and the collecting agents belonging to either its own or another domain.

Collecting Agents Server (CAS): Currently, the mobile-agents technology is based on server programs that serve as a habitat for the agents. These servers are the mediators between the agents and the system resources. The main reason for their existence is the fact that they must provide protection for the local system against the coming agents so as to avoid the harmful action of any evil agent. Since our architecture is based on mobile agents, it must necessarily have server programs that are capable of handling all the functions associated to such agents and of protecting machines where the agents live together. The CAS, or Collecting Agents Server, is the element that plays this role in our architecture and is located inside the Gatekeeper just as the RA, which was described in the previous paragraph.

This server has authorization to generate a maximum number of agents as consults arrive and then to send them. This maximum number is set according to the administrator criteria. These agents that amount to the allowed quantity are called *Permanent Collecting Agents* of the domain. We will see that this differentiation among the collecting agents has its grounds on a particular case described ahead, where the CAS can create more agents than what is allowed.

To finish the description of the CAS, we emphasize that every interaction between an external program and the collecting agents takes place through the CAS.

Gatekeeper (GK): At this level of the distributed architecture of the DSS, the GK is an abstraction of the set of programs that must be always up and running; no matter if the domain they belong to is active. In our case, the RA of the domain, the SAC and a third element must remain active. This third element has not been mentioned yet and is the Knowledge Base (BC) of the domain. These three components constitute what we call Gatekeeper and in fact, none of them is compelled to reside in the same machine where the others reside.

Router: The Router is an element that is independent of the domains and the Gatekeepers. Its function, as its name suggests it, is to give a route, but differently from the routers we are used to talking about when dealing with networks, this router must route mobile agents and not datagrams. We can point out two independent functions that must be carried out by this element. On the one hand, by saying that it must deal with mobile agents we must assume that it must have the capability of providing such agents with a safe execution environment, thus, one of the functions of this router is to act as an agent server.

On the other hand, it receives the consults of the coming agents and, according to the information in its knowledge base, gives the agents a route with the domains that can provide them with an answer.

5. Scheme of Operation of the Dynamic Consults Architecture

5.1 Creation of the Collecting Agents

When a decision-making user needs a specific type of information that he is not actually receiving and that cannot receive due to the characteristics of the contract established by his domain, he gets into the Dynamic Consults Module. There, with the help of the assistant (whose design is not within the scope of this work), he expresses the kind of information he would like to receive.

From this moment, the user gets out of the Dynamic Consults Module and knows that the system will let him know when it has an answer about the searched information. In order to know which user a consult belongs to, the latter always includes the name of the user that made it and the name of the domain where the user is registered. Thus, information will be also used for administrative purposes. The user must wait till the system lets him know that an answer is ready, since in a distributed system that operates in an asynchronous way, the domains are not compelled to remain totally active, neither within the same time and therefore a consult about a remote domain may last several hours.

Once the consult is made, the assistant passes it to the RA of the domain, which will change the consult into a format accepted by the mobile agents and will send this consult to the Collecting Agents Server (CAS).

At this moment, the CAS gives the consult to one of the permanent collecting agents of the domain or, if the allowed quantity of agents has already been sent, the CAS will store the consult and will wait till one of the agents returns. Only in case that the user has the special privilege of creating agents to serve him, the CAS will create a new agent with a certain series code and will give it the consult.

5.2 Activities of the Collecting Agents

Once the CAS has created an agent and has given it the consult, the agent goes to a special agent server: the router. This router server receives all the agents that have just left their domain, reads the formulated consult and according to the information kept in its knowledge base about the domains, it gives the agents a route of domains that can help the agent to find the searched information.

This route can contain zero to n domains. A case of zero domains is a special case where the course of action will depend on the *operation mode* of the agent. The different kinds of operation are directly related to the privileges assigned to the user that generated the consult. We can point out two groups of operation modes, where each group has two modes, and the modes of the different groups can combine. We will be describing them in detail as we go further in this work.

Group of Exploration Modalities:

- Exploration Mode
- Passive Mode

Group of Searching Modalities:

- Sequential Mode
- Parallel Mode

5.2.1 Exploration Modes

If the searching route generated by the router does not contain domains where to search for the information, we understand that the router does not have any piece of information in its knowledge base (KB) that can associate the information being searched with one or more of the existing domains. The lack of this association does not imply that the information being searched does not exist or cannot be generated. In this case, the privileges of the user that generated the consult start to play an interesting role.

If the user that made the consult has only a few privileges (among those that have the privilege to use the Dynamic Consult Module), the agent will work in the passive mode. Since it will not find domains where to collect information, it will return to the starting point and will inform the non-existence of the information being searched. This modality requires that a user with a few privileges will be able to receive results only if the information he is searching for belongs to the knowledge base of the router. Later on, we will explain how this knowledge base is brought up to date.

Another situation may occur when the agent carries a consult made by a user with many privileges. We say that may occur because a user with more privileges can also choose making a consult in passive mode. If the user makes the consult in exploration mode, the agent realizes that it has not been provided with a route in the router and sends a request to the router asking for the list of all known domains and starts visiting them searching for the information.

5.2.2 Searching Modalities

So, we have analysed how the agents know where to go so as to obtain the information they are searching for. Now we need to know the order in which they search for this information and here the second group of operation modes appears.

If the agent is working in sequential mode, it will simply go through the scales of its route as they appear. One by one, it will visit all the domains making its consult. Once it has finished and is about to go back to the domain of origin it will visit the router to inform it of the results obtained, allowing the router to have new information for bringing up to date its KB for future agents. Once it has made all this, the agent will go back to its domain, will deliver the information and will destroy itself.

If the agent is not working in sequential mode but in parallel mode, after the route of domains is obtained (if there is any), the agent will generate clones of itself, one for each of the domains to be visited (except for the one it will visit). Thus, all the clones and the original agent will be sent to their respective destinations. Each of them will travel to a domain, will make the consult to the Representative Agent of the domain, will collect the results and will go back to the router. Once all the agents, the original one and the clones, meet in the router, the original agent asks its clones, one by one, for the their information to be structured. When a clone gives the original agent its information, it tells the router the result of its incursion in the assigned domain. After that, each clone destroys itself. Once the original agent is ready to go back, it returns to the domain of origin.

5.2.3 Special Case

A special case occurs when an agent goes to a domain that is not active. Judging by what has been said till now, we could think that the agent gives up or that the router checks the activity of the domain before including the domain into the agent's route. That is not so. The fact is that although the domain may not be active, the Gatekeeper is always active and contains the RA of the domain and the CAS and from there it communicates with the RA, which will act representing the domain, that why it is called Representative Agent.

If the domain is inactive, the RA will inform the Collecting Agent of this situation and the latter will take some of the following measures. If the Collecting Agent is operating in sequential mode and has more domains to visit, it will create a clone with the consult that has to be made to that domain and will order it to remain still until the domain is activated. The original domain will go to the other domains and if any of them is still, it will repeat this operation.

When the domain activates, the RA informs this to the SAC, which will revive all the agents that are awaiting. Once the agents have made their consult, they go back to the router where the information to be delivered is structured, the knowledge base of the router is brought up to date and the clones are destroyed (just like in the case of operation in parallel mode).

If the Collecting Agent had not been working in sequential mode but in parallel mode, it would have been still until the domain was activated.

It may seem that this kind of operating gives the sequential mode an extra advantage. But we must take into account that since the domains may be in different geographic regions and with different times, the time an agent would take to visit multiple domains, waiting in each of the inactivated ones, could be increased up to several days.

6. Mobile Agents Functionalities

In order to build the interaction mechanism among DSS's domains, mobile agents involved in such mechanism should provide the following functionalities.

6.1 Basic Functions

These functions should be provided by any mobile agent.

Creation: A mobile agent is created and its state initialled.

Cloning: A mobile agent can be cloned, which means that another agent is created with the same structure and state as the original agent.

Deactivation: Mobile agents can stop their execution and store its actual state in a secondary memory. Later, it can be returned to life having the same state he had when he got deactivated.

Activation: Through activation, a mobile agent previously deactivated can be returned to life. Its state prior to its deactivation is recovered from the secondary storage.

Dispatch: A mobile agent and its state can be sent to another remote system to perform its activities.

Retract: Once an agent has been sent to a remote location, it is possible to use this function to make it come back. Once this order is received, the mobile agent returns with its current state.

Destroy: A mobile agent can be eliminated no matter where it is and which its current sate is. Its state is lost forever.

6.2 Specific functions to support interconnection mechanism among domains

In the following paragraphs we will describe additional functionalities needed to support the proposed communication mechanism among the DSS's domains.

Communication with a local RA: Agents should be able to interact with the representative agent of their domain in order to receive questions from them, and in order to let them know the answers to previous questions.

Communication with a remote RA: Agents also must be able to dialogue with a remote representative agent in order to ask it about the searched information. Also, depending on the representative agent answer, the mobile agent must decide its next operation. As an example, if the RA informs it that the domain is not active but there is certain probability that the information it is looking for is available in that domain, the mobile agent can decide to wait, to continue its search in other domains and then return, or to return to its original domain and inform that the domain that has the information is not active.

Communication with the Router: Another extra functionality is to be able to communicate with the router to ask it for a list of domains to be visited for gathering information. They must also be able to inform the results of their search to the router so that the latter can update its knowledge base.

Decision Mechanism: When the router gives an agent a list of domains, the agent must decide how it is going to search those domains. Considering its current operational mode, the agent may decide to visit all domains sequentially or to create clones to visit all domains at the same time.

Communication between clones: A mobile agent must provide the necessary functionality to communicate with its clones (if it is the original) or with its brothers (if it is a clone). This is necessary because once all clones have finished the search of the information and they meet in the router they must interact with the original one to provide him with the collected information, so that the latter can structure an answer and return to the original domain.

7. Architecture Implementation

As we have previously mentioned, our architecture is divided into two mechanisms: a contracts mechanism and a mobile agents mechanism. We will now take a look at the platforms in which these mechanisms are going to be implemented.

7.1 Contracts Mechanism

This mechanism is based on the communication existing among gatekeepers, so it suggests us the need of an appropriate communication infrastructure.

The Domains independence from the middleware is one of the distributed architecture's requirements, and this is achieved by means of the gatekeepers, which are the only components that must deal with the middleware. Because of this, we can use any middleware, like CORBA, COM, MOM, or Java RMI, and even change it at any time without affecting any domain.

In order to implement the middleware, we have decided to use the OMG CORBA standard [9], which allows distributed object computing. In our case, we use both the Event Service COSS (Common Object Specification Service) OMG and the Name Service as essential CORBA services [10][11].

The communication model of the DSS distributed architecture requires that gatekeepers use an asynchronous, uncoupled and multi-cast communication paradigm. This means that the sender of a message is not compelled to wait for an answer, that objects communicate which each other through the use of mediators so they do not have to know each other, and that a message from an object can be sent to multiple receivers. We have implemented this by using the Event service COSS OMG based on the CORBA standard [12], which specifies three primary components: providers, consumers and the

event channel. The event channel is a standard CORBA object, that allows asynchronous and uncoupled communication among multiple providers and multiple consumers.

We finish the explanation about the implementation of the contracts-based mechanism saying that all domains have been coded using an Object Oriented programming language [13].

7.2 Mobile Agents Mechanism

As regards mobile agent mechanism, we can say that this part of the architecture is being programmed using the Java language, which offers not only a multi-platform environment, but also a very robust environment when something goes wrong.

In the particular case of mobile agents, we are experimenting with a prototype version created using the "Aglets Software Development Kit" also known as "ASDK", created by IBM's laboratories in Tokyo using the Java language [14].

One of the motivating factors in the election of a framework for mobile agent creation, was the necessity of a framework not only capable of allowing the creation of mobile agents, but a framework also capable of allowing the creation of server programs.

Besides, this development kit includes an application programming interface (API) with security control functionality (SecurityAPI) by means of the definition of authorities, privileges and preferences. Even more important, the ASDK framework implements the MASIF interface, although it does not complies with the standard because it does not uses CORBA yet [15].

Inside Aglets architecture, agents have the ability to intercommunicate through the use of a proxy related to each agent. The proxy brings location transparency and protects him from other agents trying to call its methods. There are classes for encapsulating messages passed among agents and there are methods for handling these messages.

Lastly, we can say that the functions to handle the events in which a mobile agent incurs are perfectly identified.

8. Conclusions

We have designed a DSS distributed architecture that allows the domains that support the decision processes of an industrial organization, to work in a cooperative way. The DSS distributed architecture allows the interaction among applications of different domains with a low coupling level.

The event oriented communication model with the publication/subscription paradigm on which the contracts mechanism of the distributed architecture is based allows the DSS to support the communication among domains in such a way that it can be adapted to the business process needs of the organization.

The DSS distributed architecture mechanism derived from the federated architecture satisfies the quality attributes (scalability, modifiability, portability, reusability and adaptability) of the federated architecture (Wijegunaratne, 1998). We have used the CORBA technology of OMG in order to implement this mechanism of the distributed architecture. This middleware provides appropriate high level tools to specify object oriented interfaces. These tools allowed us to define the contract among domains and to encapsulate them. On the other hand, this middleware also allowed us to build an asynchronous and uncoupled communication mechanism among domains called publication/ subscription.

On the other hand, the mobile agents-based mechanism of the distributed architecture allows the domains to carry out searches and gathering of information that are not included in the contracts of the previous mechanism. That is, through the mobile agents mechanism, any domain can search any information type that was not foreseen at the design time. In this way the distributed architecture provides a flexible mechanism to support the information requirement of dynamic decision processes.

Finally, our implementation of the mobile agent mechanism is a preliminary prototype, and therefore we can only say that it promises to be viable. We are working to implement a more complete prototype to be used for performance analysis of the mobile agent mechanism for the interaction among domains to support dynamic decision processes.

References

1. Rico, O.Yuschak, M.L.Taverna, J.Ramos, M.R.Galli y O.Chiotti, (1997) *DSS Generator for Industrial Companies*, Com. and Ind. Engng, Vol 33, 1-2, pp. 357,360.
2. Wijegunaratne, I. and G. Fernández, *Distributed Applications Engng*, Springler, 1998.
3. Villarreal, P, A.Toffolo, J. Nagel, P. Rossi, O.Chiotti, *SSD Global Distribuido*, CLEI, Conferencia Latinoamericana de Informatica, Vol 1, pp 87-98, 1999.
4. Nagel, J., P.Rossi, A.Toffolo, P.Villarreal, and O. Chiotti *Distributed Architecture of a Global DSS Generator for Industrial Companies*, Proceeding 25th International Conference on Computers and Industrial Engineering, Lousiana,USA, 1999.
5. Malik, N.A. (1996), The three-tier Client/Server Model, in Bettone, G. El al (eds.) Tricks of the visual Basic 4 Gurus, SAMS Publishing.
6. James E. White, General Magic Inc., 1997 , *Mobile Agents*, in Jeffrey Bradshaw's book *Software Agents*, AAAI Press/The MIT Press 1997.
7. David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik, *Itinerant Agents for Mobile Computing,* in Michael Huhns' & Munidar Singh's book, *Reading in Agents, 1998*.
8. Daniela Rus, Robert Gray, and David Kots, Department of Computer Science, Dartmouth College, *Transportable Information Agents*, in Michael Huhns & Munidar Singh book, *Reading in Agents, 1998*.
9. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.0 ed., Julio 1995.
10. Schmidt D. C. y S. Vinoski, *The OMG Events Service (Column 9), C++ Report, vol. 9*, Febrero 1997.
11. Object Management Group, *CORBAServices: Common Object Services Specification, Revised Edition*, 95-3-31 ed., 1995.
12. CORBAplus for C++, CORBAplus event service and CORBAplus naming service user's guide, (1997)
13. DELPHI 3 user's guide, (1997).
14. Danny B. Lange, IBM Tokyo Research Laboratory, Feb. 1997, *Java Aglet Application Programming Interface (J-AAPI) White Paper - Draft 2* http://www.trl.ibm.co.jp/aglets/JAAPI-whitepaper.html

Amund Aarsten, Davide Brugali, Giuseppe Menga, Dept. of Automatica e Informatica, Politecnico di Torino, *Architectural Aspects of Agent-Based Systems*