A Holonic Approach to Model and Deploy Large Scale Simulations

Sebastian Rodriguez, Vincent Hilaire, and Abder Koukam

Université de Technologie de Belfort Montbéliard,
90010 Belfort Cedex, France
sebastian.rodriguez@utbm.fr
Tel.: + (33) 384 583 009

Abstract. Multi-Agent Based Simulations (MABS) for real-world problems may require a large number of agents. A possible solution is to distribute the simulation in multiple machines. Thus, we are forced to consider how Large Scale MABS can be deployed in order to have an efficient system. Even more, we need to consider how to cluster those agents in the different execution servers. In this paper we propose an approach based on a holonic model for the construction and update of clusters of agents. We also present two modules to facilitate the deployment and control of distributed simulations.

1 Introduction

Multi-Agent Based Simulations (MABS in the sequel) are based upon the analogy between real world entities and autonomous and interacting agents. This is a natural and intuitive approach for problem simulation.

However, for real world problems, MABS frequently leads to a great number of agents. Any MAS platform, such as FIPA-OS[18], JADE[1] or MadKit[14], inherits operating system and hardware layers constraints e.g. memory, cpu and thread number limits.

In this context we are forced to consider how Large Scale MABS can be deployed in order to have an efficient system. A possible solution is to profit from MAS' intrinsic decentralized nature to distribute agents on several computers.

Two issues have then to be considered. First, how to group agents that will execute in the same machine. And second, how do we deploy and control a distributed system or simulation.

In order to support the decentralized characteristic of the agent paradigm, we need to provide the MAS with a platform that enables distributed interactions in a transparent way from the agent's point of view. To face this problem, we propose two plugins for the MadKit platform [14] that facilitate MAS deployment and the control of a distributed simulation.

We also discuss the issue of how to create the clusters of agents that will be sent to the machines. Indeed, before distributing the MAS, we need to find means to create clusters of agents that will execute in the same machine. To

L. Antunes and K. Takadama (Eds.): MABS 2006, LNAI 4442, pp. 112–127, 2007.

tackle this problem we define a distribution logic using the holonic paradigm. This distribution follows the holon organizational structure.

Holons were defined by Koestler [15] as self-similar structures composed of holons as substructures. They are neither parts nor wholes in an absolute sense. The organizational structure defined by holons, called holarchy, allows the modelling at several granularity levels. Each level corresponds to a group of interacting holons.

The paper is organized as follows: section 2 presents our approach for holonic modelling and illustrates it with the traffic simulation model of a big plant. Section 2 presents the principles of the MadKit platform and the two plugins we have created to handle large scale MABS. Section 4 details the plant traffic simulation and section 5 concludes.

2 Holonic Modelling of Large Scale Simulations

One of the issues discussed in this paper is: how do we create the clusters of agents that will be sent to the machines?

In our approach we will base that decision on the holarchy. In the next section we briefly introduce our framework for holonic MAS. For a complete definition of this framework see [22,23]. [21] presents an approach for environment modelling using this framework and in [20] we have proven pertinent properties about this framework.

2.1 Holonic Framework

Holonic MAS have attracked recently much interest of the DAI community. These types of systems have been applied to a wide range of domains such as Transports [3], Manufacturing Systems [25,16], etc.

However most of the frameworks proposed to model them are strongly related to their domain of application. This renders the approach sometimes difficult to apply to other problems. In an attempt to solve this drawback, we based our framework in an organizational approach [23]. The framework uses then organizations to model the status of the members (sub-holons) in the composition of higher level holons (super-holons) and to model the interactions of the members to achieve their goals/tasks.

We have adopted a moderated group structure for holonic MAS[12]. This decision is based on the wide range of configurations that are possible by modifying the commitments of the members toward their super-holon. In a moderated group, we can differentiate two status for the members. First, the *moderator* or *representative*, who acts as the interface with non-member holons, and, second, *represented* members, who are masked to the outside world by their representatives. Even if we use the name "*Moderated Group*" for compatibility with earlier works in this domain, it can be misleading. As we see it, the structure does not necessarily introduced any authority or subordination. The name makes reference to the different status found in the group. We can then adapt this

organization by giving the representatives specific authorities according to the problem or constraints.

In order to represent a moderated group as an organization we have identified a set of roles that can represent these concepts. We have chosen to use four roles to describe a moderated group as an organization: Head, Part, Multi-Part and StandAlone. The three first roles describe a status of a member inside a super-holon. The Stand-Alone role represents, on the other hand, how non-members are seen by an existing holon.

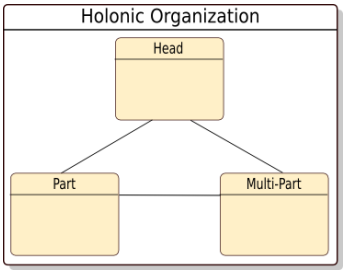

Fig. 1. RIO Diagram of the holons members

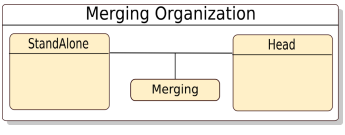

Fig. 2. RIO Diagram of the Merging Organization

As shown in the figure 1 the representatives of the super-holon play the *Head* role. A *Head* member becomes then part of the visible face of the super-holon. This means that the *head* becomes a kind of interface between the members of the holon and the outside world. The *head* role can be played by more than one member at the same time.

The members can confer the *head* a number of rights or authorities. According to the level of authority given to *heads*, super-holon can adopt different configurations. Thus, the *Head* role represents a privileged status in the super-holon. *Heads* will generally be conferred with a certain level of authority. However, these members have also an administrative load. This load can be variable depending on the selected configuration.

It is important to remark that when a set of holons merge into a super-holon a new entity appears in the system. In this case, they are not merely a group of holon in interaction as in "traditional" MAS theory. The super-holon is then an entity of its own right. Thus, it has a set of skills, is capable of taking roles, etc. At the same time, as *Heads* constitute the interface of the super-holon, they are in charge of redistributing the information arriving from the outside. And, thus to "trigger" the (internal) process that will produce the desired result. The *Part* role identifies members of a single holon. These members are represented by *Head*s within the outside world. While the holon belongs to a single super-holon, it will play this role. However, when the holon is not satisfied with its current super-holon it has two possibilities. The first is to quit its super-holon entirely and try to find a new holon to merge and collaborate with. The second

is to try to merge with a second super-holon while remaining as a member of the first super-holon. In this case the holon will change his role to $Multi - Part$. The $Multi - Part$ role is an extension of the $Part$ role. It puts emphasis on a particular situation when a sub-holon is shared by more than one super-holon.

In order to support the integration of new member, we need to provide external holons with a "standard" interface so they can request their admition. From the super-holons point of view, external holons are seen as *StandAlone* role players. When a super-holon is created, only $Heads$ belong to the interface of the super-holon. Thus, other members ($Part$ and $MultiPart$) should not be visible by external holons. This is modeled by the organization presented in figure 2. In this organization, *StandAlone* holons may interact only with the *heads* of the super-holon.

2.2 Holarchy Example

In order to illustrate our framework we take an example and describe it with holonic concepts. This example consists in a simplified University. Imagine that we model the university as composed of Departments and research Laboratories. They are in turn composed of Professors and Researchers respectively. If we isolate the Computer Science and Laboratory Holon and their components from the university example and we add these holonic roles, we obtain figure 3. $Part$ role players for the laboratory represent researcher that belong only to the laboratory, e.g. full time researchers. On the other hand, some researcher may, in addition to their activities in the laboratory, give lectures in the computer science department. These holons, like holon RP in figure 3, belong to both super-holons simultaneously and thus they play the $MultiPart$ role. In this example, the department and laboratory directors would be the $Heads$ of the C.S. Department and the laboratory respectively.

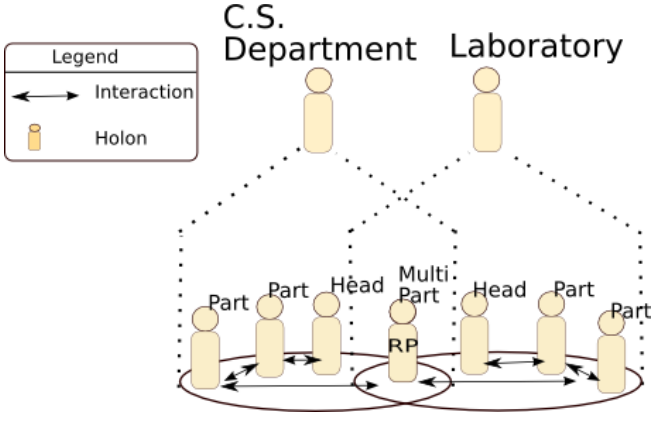

Fig. 3. Department and Laboratory Holons

As we mentioned earlier other organization will be used to specify domain dependant interactions (e.g. a *Lecture Organization* to describe how professors interact with their students) [23].

Based on these *holonic* roles –Head, Part and MultiPart– we have defined mechanisms to handle holons dynamics. They are based upon the affinity and satisfaction between holons. The notion of *Affinity* was inspired by a technique used for the Artificial Immune System [6]. The term *Satisfaction* has often been used to represent the gratification of an agent concerning its current state or the progress of its goals/tasks [5,24].

The affinity between holons must be defined according to the domain of the application. The affinity measures, according to the application's objectives, the compatibility of two holons to work together toward a shared objective.

The *compatibility* of two holons means that they can provide help to each other to progress towards their goals. Based on the application's objective, we define a set of rules that allow us to evaluate this *compatibility*. Generally speaking, we can say that two holons are compatible if they have shared goals and complementary services.

Using these two notions, holons are able to decide when they should join or leave a super-holon (satisfaction) and with which super-holon to merge with (affinity). Holons can then move from one super-holon to another as the system (in our case simulation) evolves.

2.3 PSA Simulation Model

We propose the use of holarchies for the modelling of simulation environments. In the Peugeot SA (PSA in the sequel) plant example we want to simulate the traffic within the plant. The environment of this simulation is defined by the topology of the plant. The agents will be the different vehicles driving through the plant.

The environment will be represented by a holarchy. This holarchy defines the organizational and topological structure in which agents will evolve. Each environmental holon will enforce contextual physical laws and represent a specific granularity level of the real plant topology. This holarchy is predefined as it represents the real plant environment. Indeed, the latter cannot change and the physical laws we need to enforce are known a priori. In order to represent the geographical environment of the plant as a holarchy we have to find recursive concepts which represent the plant's environment parts. The concepts we have chosen are described in the figures 4 and 5.

Figure 4 shows that a road is divided into links. A link represents a one way lane of a road. A segment is composed of, at least, two exchange points, called input and output exchange points, and a link. Exchange points let vehicles pass from one link to the other. An exchange point is always shared by at least two segments.

In the figure 5 we can see that the industrial plant is composed of a set of zones, that in turn contain Buildings and Segments. Buildings and Segments can also communicate through shared exchange points. Usually an exchange point

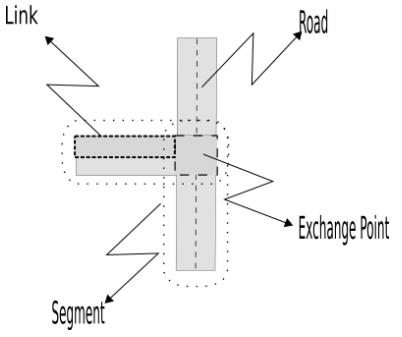

Fig. 4. Roads, Segments, Links and Exchange Points

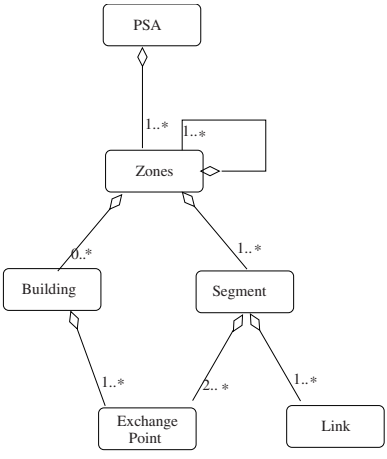

Fig. 5. Conceptual view of the plant

represents a crossroad, but in can also represent an entrance used by trucks to access buildings. A zone may also be decomposed in smaller zones which contains Buildings and Segments and so on recursively.

This decomposition also gives us important information about the roles involved. Let us consider the *ExchangePoint* role. This role represents an exchange point between physical entities such as roads and buildings. An ExchangePoint can then be specialized to respect certain constraints, for example, a door lets a human get into a building but it's impossible for a car to use it. As we can see the exchange point is a "special" role from the "holonic point of view" since the role is actually shared by more than one holon by definition. Note that this situation differs from the one where a particular holon plays the multi-part role. In this case, we know that the holon preforming this role will be shared by at least two holons prior to the simulation, even more, we can even know exactly to which two super-holons. Such a hierarchical decomposition of the environment presents several advantages when compared to a global representation. First, no size limit is imposed by the model, this enables us to use the same environment decomposition to simulate the traffic inside a city or a (much) smaller industrial plant. Some semantical information could be introduced, like this, instead of zones, we will represent quarter, block, etc. [7].

Second, all necessary information to simulate the traffic inside a link is local (other vehicles, roadsigns, etc). This makes the model easier to distribute in a network and leaves the door open to Real-time applications as well as Virtual Reality implementations.

It is important to notice that the decomposition may continue in order to provide a higher level of detail. It provides a simple way of decomposing different types of environments. For instance, a building itself can be decomposed in Rooms and Exchange Points(doors).

On the other hand, this type of decomposition imposes a highly hierarchical and decentralized representation of the environment. This could present some disadvantages when the environment presents some global "variables" accessible for all agents.

In our model, the vehicle agent is able to change a set of variable that affect the vehicle's state. These variables are later used by the environment to adjust the vehicle's speed according to the environmental principles and rules. Vehicles can query their current link to obtain information about road signs, traffic lights, maximal speed limits, etc. They can also request information about adjacent link to the exchange points.

3 Modules for Large Simulations

The model presented in the previous section considers how to cluster agents into coherent groups according to the application. However, it assumes that all holons in the system can communicate regardless of their physical location. To enable this behavior, we propose two plugins for the MadKit platform. An overview is presented in figure 6. The first plugin (NetComm), presented in section 3.2, is in charge of generating and maintaining the "Virtual Community" between kernels. On top of that community the second plugin (SimSever), presented in section 3.3, offers the possibility to distribute and control a simulation.

Before we detail the architecture of these plugins, let us first briefly review MadKit 's principles.

3.1 MadKit Principles

MadKit is built upon the AGR model [8] illustrated in figure 7. This model is based on the following organizational concepts: Agent, Group and Role. An

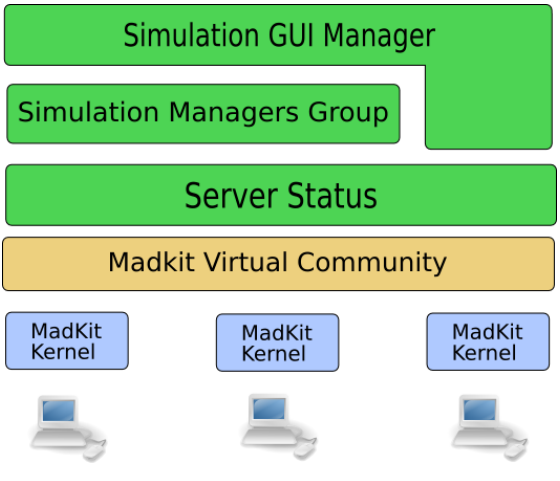

Fig. 6. Plugins Overview

agent is an active, communicating entity playing *roles* within *groups*. An agent may hold multiple roles, and may be a member of several groups. An important characteristic of the AGR model is that no constraints are placed upon the architecture of an agent or about its mental capabilities. Thus, an agent may be reactive as an ant, or deliberative with mental states.

A group is a set of agents sharing some common characteristics. A group is used as a context for a pattern of activities. Two agents may communicate if and only if they belong to the same group, but an agent may belong to several groups.

A role is the abstract representation of a functional position of an agent in a group. An agent must play at least a role in a group, but an agent may play several roles. Roles are local to groups, and a role must be requested by an agent. A role may be played by several agents.

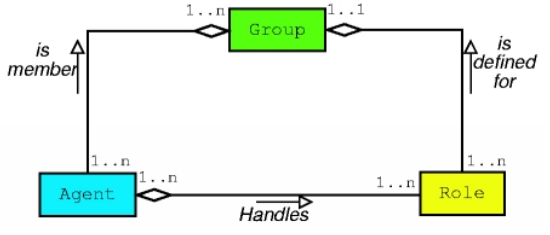

Fig. 7. AGR Model from [13]

The MadKit platform proposes libraries to create/join groups, take roles, send messages to other agents via the roles they play, etc. It is written in JAVA using the micro-kernel principle. A MadKit kernel is created before agents are launched and intercept all service calls.

3.2 Transparent Connection of Kernels

One interesting characteristic of the MadKit platform is that, from the agent's perspective, there is absolutely no difference between the communication with local agents and the communication with distant agents.

Even if MadKit offered a plugin, called Communicator, to interconnect kernels, it presented a few disadvantages. Among them, the Communicator offers a single "hard coded" protocol for the communication between kernels. This approach restricts the network capabilities evolutions of the platform. It also required to know a priori the distant kernel's address and port.

To tackle these problems, we developed a new plugin for MadKit called NetComm. This plugin presents a multi-agent design, allowing different protocol to be used and featuring an automatic detection and connection of existing kernels in the network.

The underlying idea of the NetComm Module is to have a group of agents that will manage all incoming and outgoing communication. Each foreign kernel will interact with one local agent in commonly selected protocol. This approach lets us envisage a number of different protocols, that will be selected according to the situation.

Following the MadKit architecture, a specific agent must register as the Communicator, or Communication Responsible. In NetComm this agent is the NetAgent. The NetAgent is a sort of representative of all other agents in the communications module. We now briefly describe the main agents present in the module. Figure 8 shows the general structure of the NetComm Plugin.

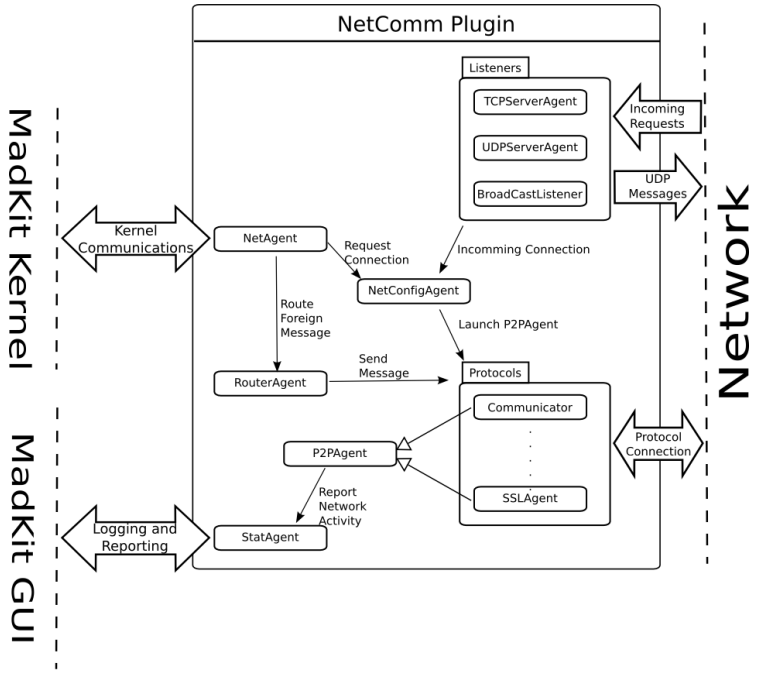

Fig. 8. NetComm Overview

The **NetAgent** can be called the main agent of Netcomm. This agent will represent the whole communication system in front of the local kernel.

The **RouterAgent** agent takes care of routing the messages received from the NetAgent to the P2PAgent responsible of the connection with the concerned Kernel.

The **NetConfig** agent is launched every time a new kernel is going to be connected. First the agent will try to know whether at the other end there is a `CommunicatorAgent` (the original communication module of MadKit) or another NetConfigAgent. If the agent is interacting with a Communicator agent,

an instance of the Communicator compatible agent will be launched. When interacting with a NetConfigAgent, a protocol is followed to establish a common protocol. If a common protocol is found, it will be sent and the corresponding P2PAgent will be launched.

The **P2PAgent** is the super class of the agents responsible for the connections with other kernels. Several types of P2PAgents exist like the SSLAgent enabling SSL communications.

The **StatAgent** keeps track of the network usage statistics. The statistics can be enabled or disabled in real-time by sending a NetConfigMessage. In the time being the StatAgent reports only through its graphical interface. However future work will enable this agent to log the network traffic in a file for later analysis.

Three different agents, the **Listeners**, are in charge of listening incoming requests, one per used protocol. Thus we have a TCP, UDP and Broadcast Listeners. Incoming request will start a new NetConfigAgent to configure the connection with the foreign kernel.

This plugin is concerned with the discovery of new kernels in the network and provides the *Virtual* MadKit *Community* that we need to deploy our agents. It is important to notice, that the network may grow at runtime. Indeed, new kernels can be integrated to the community dynamically, since MadKit 's kernels can synchronize their information.

3.3 Distribution of a Simulation

The NetComm plugin described previously gives us the possibility to consider that all kernels in the system are capable of automatically connecting, thus providing a unique virtual MadKit Community. The second problem to tackle is the distribution of the agents themselves. A second module was developed for this purpose, called *SimServer*. In distinction with NetComm, it does not intend to be a generic plugin for any simulation. This plugin tries to reduce the development time to distribute a simulation for all those that do not need to introduce a migration mechanism into agents. Even if we used the NetComm plugin to interconnect the kernels, we could use any other means to connect the kernel, the communicator or a third module. This is to say that the SimServer plugin is completely independent of the NetComm plugin. In top of the MadKit "Virtual Community" a number of groups are created to control and observe the simulation. The first is the "Simulation Status". This group aims to provide statistics of the state of the servers in terms of memory, cpu load, etc. The organization consists of three roles: *StatusManager*,*StatusRequester* and *ServerInformer*. The **ServerInformer** is in charge of collecting the information of the server where it is running and informs the Status Manager. This information contains, but is not limited to, memory, cpu load, simulations running in the server and agents per simulation.

The **StatusManager** collects the information and formats it for logs and special request.

The **StatusRequester** role is played by agents willing to get information about a simulation or the status of the servers. It requests the information to the StatusManager. For instance, a simulator will typically have the possibility to show to the user information about servers where the user's simulation is running on. For this, an agent should play the StatusRequester roles to get the required information.

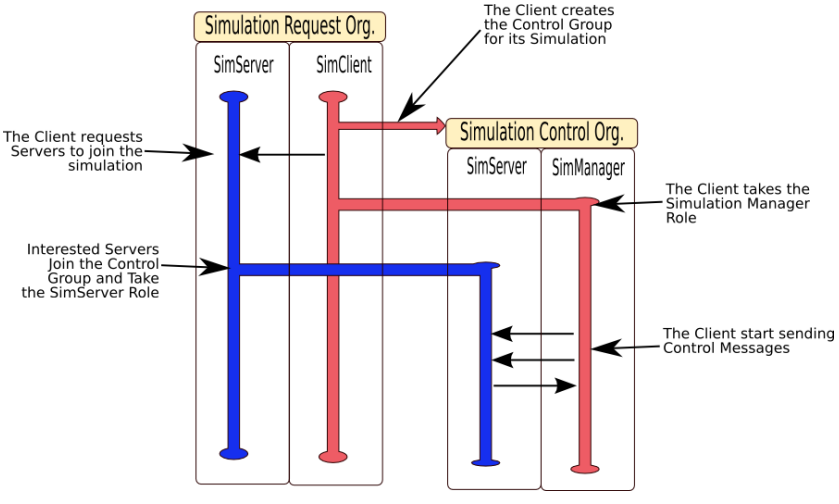

Fig. 9. Sequence diagram used to create a new Simulation

When a new simulation needs to be created a specific protocol must be followed. An *organizational sequence diagram* (this type of diagrams was presented in [9]) illustrates this protocol in figure 9.

1. An agent takes the *SimulationClient* role. This agent creates a group for the simulation. This group will be used to control the simulation itself.
2. The client broadcasts a message to all *SimulationServer* role players. This message informs the servers that a new Simulation needs to be created. The message contains information about requirements of the simulation, e.g. number of agents to be created, name of the group to control the simulation, etc.
3. Servers interested in participating in the simulation join the simulation management group created in step 1. This decision is based on the status of the server it represents, i.e. cpu and memory load, number of agent already running in the server, etc.
4. The SimulationManager (role of the Control group) distributes the required information to instantiate the agent required for the simulation, e.g. the agents identifiers, classes to load, etc.

5. Once all the servers acknowledge that they are ready to start, the SimulationManager may start sending Control messages to the server in its group, i.e. start, stop, etc.

The module contains a default implementation of the agents that provide a basic support. These agents are a generic implementation and make a number of assumptions. However, they can be used as starting point to develop a more suitable implementation, in particular for the simulation controllers.

4 Simulation

As presented in section 2.3, the environment is modeled as a holarchy. Each holon of this holarchy represents a specific context. For the PSA example it's a specific place in the plant. These places have different granularity levels according to their level in the holarchy. During the simulation, vehicle agents move from one holon to another and the granularity is chosen by execution or simulation constraints such as which features can be observed.

The dynamic choice of the environment granularity level during the simulation must be transparent for the agents. In order to do this, agents use our holonic framework and specifically *ExchangePoint* holons which enable the communication between holons of the same level and connected in the plant's topology. The figure 10 describes the sequence of messages exchanged between the *ExchangePoint*, a vehicle and the Segment's Head. The vehicle agent is moving along the segment 1 and requests the exchange point to forward a merging request. The exchange point forwards the request and receives a reply. The reply is forwarded to the vehicle. If the reply is positive the vehicle can merge with the segment 2 holon as shown in figure 11. These interaction sequences are a mean to represent the influence/reaction model [10]. Indeed, the agent emits influences in asking to merge with a specific holon. The environment is able to determine the eventual answer according to jams or environment properties.

Notice that using this mechanism does not require the environmental holons to execute in the same kernel (or machine). Indeed, the virtual community, created using the NetComm plugin, enables holons to communicate transparently. When vehicle holons move between segments, they also move to the segment's execution kernel. Like this, most frequent interactions (e.g. between the link and the vehicle and between vehicles in the same link) will always be executed locally. So, clusters of agents are created on different machines following the structure of the holarchy.

The whole simulation can be controlled and observered using the SimServer Plugin.

This approach enables one to describe the environment with multiple levels of granularity; examples are given in figures 12 and 13. In figure 12 we can view the simulation of several roads, crossroads and buildings. The figure 13 is a more fine grain simulation of a crossroad. Nevertheless the simulation of the rest of the plant is always running in the two cases.

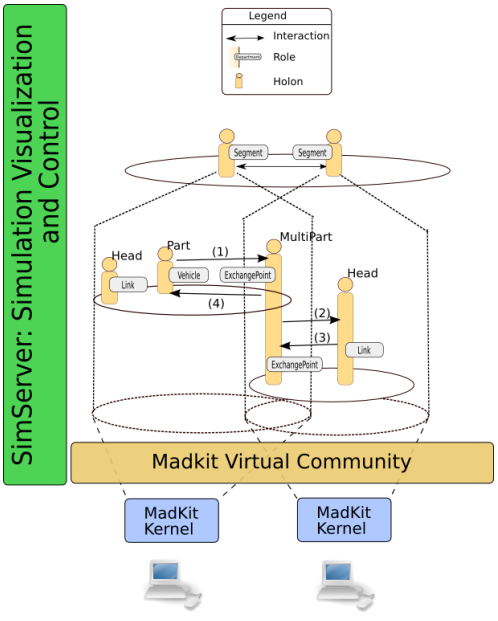

Fig. 10. Access request sequence

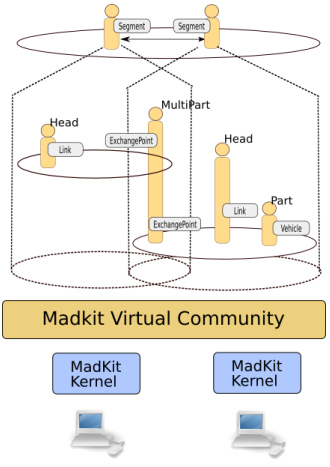

Fig. 11. If accepted, the vehicle moves to the next segment

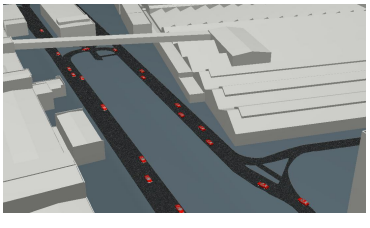

Fig. 12. View of different crossroads and buildings

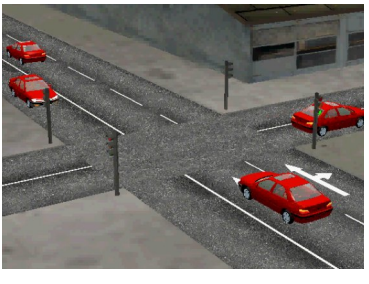

Fig. 13. Crossroad close up

5 Related Works

Two problems are frequently encountered during MABS [2]. First, how to model the agents and their environment to exploit inherent agents parallelism. Second, how to choose an appropriate grain size for the simulation. These two problems are related. Indeed, chosing an inappropriate grain size could hamper the distribution of the simulation [2].

In the sequel we compare our work with existing approaches combining concepts for the conceptual distribution of MABS with implementation support.

The PDES-MAS project approach [2] consists in partitioning the environment in zones called "interest areas" or "spheres of influence" concept similar to holons in our case. It is a specific locus of interaction and perception for agents. These clusters execute concurrently. The drawback of this approach is that it doesnt take into account multiple granularities.

The MACE3J architecture [11] is based on two modelling hypothesis. First, there exists an organizational structure called ActivationGroup which is a group of agents acting in relation with each another. This structure is implemented by an object which contains services such as: scheduler, time manager, environment, ... Second, agents havent any imposed architecture but must implement the Registerable JAVA interface which allows request from the ActivationGroup. These agents group result in a flat architecture such as the one use within the MadKit platform [14].

The MadKit platform [14] proposes mechanisms which ease the deployment of distributed simulations. We have already discussed in the section 3 the drawbacks of these mechanisms. The conceptual model of MadKit , namely AGR, does not propose any clustering concept except the organization. It may be difficult to take into account efficiency problems of a distribution only based upon organizations.

The RePast [4] agent simulation toolkit, which uses the ideas of the well-known SWARM platform [17], offers services to display and monitor simulations. To our knowledge there is no facility proposed for the distribution of simulations. The SPADES system [19] propose an architecture for deploying parallel or distributed simulations based on a discrete simulation engine. The agents must be based on a sense/think/act loop which is not a strong assumption. The drawback of this approach is that it supposes that events are centralized in a master process which has complete knowledge of all pending events. This hypothesis may hinder the scalability of MABS.

6 Conclusion

In this paper we have presented an approach to model and deploy large scale and variable scale MABS. This approach is based upon holonic MAS and is supported by two plugins which ease the distribution, monitoring and control of simulations hence reducing the complexity of deploying distributed MABS.

The distribution and clustering of agents follows the holarchy structure and thus reduces distant message passing. This issue is frequently discussed in approaches for distributed MABS. Each holon defines a cluster of agents executing on a same computer or part of a network. These agents share the same part of the environment. The interactions are thus mostly locals. Even more, agents may change their executing machine as they move in the environment. The distribution also allows the connection of the simulation to visualization tools such as Virtools as it is the case in figures 12 and 13.

The first plugin, NetComm, was designed to allow an automatic connection of the MadKit kernels in a network. These connections allow the creation of a virtual community so that the distribution of kernels is transparent. NetComm also enables the creation of new kernels and or the relocation of existing kernels. The second plugin, SimServer, enables the distribution of agents, the management and monitoring of the simulation. By distribution of agents we mean that an agent must belong to a specific kernel which may change during its lifetime. This kernel change is illustrated by an example in figures 10 and 11. Management and monitoring facilities are provided in order to control the overall simulation and to be able to visualize the simulation results. Both are plugins for the MadKit platform. These modules allows the execution of a great number of agents. Among the future works, we plan the integration of analytic tools in the plugins.

References

1. Bellifemine, F., Poggi, A., Rimassa, G.: Jade - a fipa-compliant agent framework. Technical report, CSELT (1999)
2. Logan, B., Theodoropoulos, G.: The Distributed Simulation of Multi-Agent Systems. Proceedings of the IEEE 89 (2001)
3. Bürckert, H.-J., Fischer, K., Vierke, G.: Transportation scheduling with holonic mas - the teletruck approach. In: Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents, pp. 577–590 (1998)
4. Collier, N.: Repast: Recursive porus agent simulation toolkit
5. Christine, K., Fernandes, C.e S.: Systèmes Multi-Agents Hybrides: Une Approche pour la Conception de Systèmes Complexes. PhD thesis, Université Joseph Fourier-Grenoble 1 (2001)
6. Dasgupta, D.: Artificial Immune Systems and Their Applications. Springer, Heidelberg (1998)
7. Farenc, N., Boulic, R., Thalmann, D.: An informed environment dedicated to the simulation of virtual humans in urban context. In: Proc. Eurographics '99, pp. 309–318, Milano, Italy (1999)
8. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Demazeau, Y., Durfee, E., Jennings, N.R. (eds.) ICMAS'98 (July 1998)
9. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) Agent-Oriented Software Engineering IV. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
10. Ferber, J., Müller, J.-P.: Influences and reaction: a model of situated multiagent systems. In: ICMAS'96 (December 1996)
11. Gasser, L., Kakugawa, K.: Mace3J: fast flexible distributed simulation of large, large-grain multi-agent systems. In: AAMAS, pp. 745–752. ACM, New York (2002)
12. Gerber, C., Siekmann, J.H., Vierke, G.: Holonic multi-agent systems. Technical Report DFKI-RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz - GmbH, Postfach 20 80, 67608 Kaiserslautern, FRG (May 1999)
13. Gutknecht, O.: Proposition d'un Modéle Organisationnel générique de systéme multi-agent. Examen de ses conséquences formelles, implémentatoires et méthodologiques PhD thesis, Université de Montpellier II (September 2001)

14. Gutknecht, O., Ferber, J., Michel, F.: Madkit: Une expérience d'architecture de plate-forme multi-agent générique. In: 8iémes Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systémes Multi-Agents (JFIADSMA'2000), Saint Jean La Vêtre, France (2000)
15. Koestler, A.: The Ghost in the Machine. Hutchinson (1967)
16. Maturana, F., Shen, W., Norrie, D.: Metamorph: An adaptive agent-based architecture for intelligent manufacturing. International Journal of Production Research 37(10), 2159–2174 (1999)
17. Minor, N., Burkhart, R., Langton, C.: The swarm simulation system: A toolkit for building multi-agent simulations (1996)
18. Poslad, S., Buckle, P., Hadingham, R.: The fipa-os agent platform: Open source for open standards. In: Proc. of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, pp. 355–368 (2000)
19. Riley, P.: SPADES A System for parallel-agent, discrete-event simulation. AI Magazine 24(2), 41–42 (2003)
20. Rodriguez, S., Hilaire, V., Koukam, A.: Fomal specification of holonic multi-agent system framework. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J.J. (eds.) ICCS 2005. LNCS, vol. 3516, pp. 719–726. Springer, Heidelberg (2005)
21. Rodriguez, S., Hilaire, V., Koukam, A.: Holonic modelling of environments for situated multi-agent systems. In: Second International Workshop on Environments for Multiagent Systems @ AAMAS 2005, pp. 133–138 (2005)
22. Rodriguez, S., Hilaire, V., Koukam, A.: Towards a methodological framework for holonic multi-agent systems. In: Fourth International Workshop of Engineering Societies in the Agents World, pp. 179–185, Imperial College London, UK (EU) (2003)
23. Rodriguez, S.A.: From analysis to design of Holonic Multi-Agent Systems: a Framework, methodological guidelines and applications. PhD thesis, Université de Technologie de Belfort-Montbéliard (2005)
24. Simonin, O., Ferber, J.: Modélisation des satisfactions personnelle et interactive d'agents situés coopératifs. In: JFIADSMA'01: 9èmes Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents, pp. 215–226 (2001)
25. Wyns, J.: Reference architecture for Holonic Manufacturing Systems - the key to support evolution and reconfiguration. PhD thesis, Katholieke Universiteit Leuven (1999)