A OO Framework for Multi-Agent Systems

S.Bandini, F. De Paoli, S. Manzoni
DISCO - Università di Milano Bicocca
Via Bicocca degli Arcimboli 8, 20126, Milano
{bandini, depaoli, manzoni}@disco.unimib.it

C. Simone
DI - Università di Torino
Corso Svizzera 185, 10149, Torino
simone@di.unito.it

Abstract

*Multi-Agent Systems (MAS) are of great interest as they can be used to model complex phenomena that involve several actors that interact in various ways. This paper describes a framework to support the development of Multi-Agent Systems. The framework is based on the Reaction Diffusion Machine (RDM) and defines the run-time support for the L*MASS language. The language formalizes the concepts of the framework to provide designers with constructs that facilitate the development of Multi-Agent Systems. The underlying RDM ensures soundness to both language and framework. The work is still in progress; therefore, what is presented in this paper need to be further discussed and consolidated.*

1 Introduction

Many different definitions have been given for *Agent-Based Systems*. Usually all the definitions agree on the fact that an agent is a real or virtual entity endowed with autonomy, social ability, reactivity, and proactivity [1]. An agent can be thus described as an entity able to perceive the surrounding environment through sensors, and acting through actuators according to the perceptions. Agents can be classified as cognitive or reactive [2]. Reactive agents are elementary agents without memory and with a defined position in time and space. Cognitive agents, instead, behave in a more complex way, and their actions are based also on past experience. Whereas cognitive agents, for every possible sequence of perceptions, act to maximize a given utility function [3], reactive agents perform their actions in consequence of the perception of stimuli coming either from other agents or from the environment. In this case, the motivation for an action comes from *outside*, rather than from *inside*. As a consequence, a Multi-Agent System (MAS) can be composed of cognitive agents (generally a low number of intelligent agents), each one with a specific knowledge that determines its behavior and its interactions with other agents and the environment. By contrast, there could be MASs composed of reactive agents. This type of system is based on the idea that it is not necessary to include individually intelligent agents in the system to exhibit complex (intelligent) behaviors. Moreover, systems of reactive agents are usually more robust and fault tolerant than other agent-based systems, e.g. the loss of an agent does not have catastrophic effects for the whole system. Other benefits include flexibility and adaptability in contrast to the inflexibility of systems of cognitive agents.

Recently, the evolution of Multi-Agent Systems and the growing interest in multi-agent development platforms have leaded to some interesting tools for agent software developers. AgentBuilder [4], JACK [5], JADE [6], JAFMAS [7], Madkit [8] and Zeus [9] are some examples of this type of tools. Globally platforms for development of agents are widely heterogeneous. Although, some platforms are grounded on well-known models, like BDI model [10] for agent architecture or KQML [11] and FIPA ACL [12] infrastructures for communication among agents, they limit the development of MASs designed according to these models. Some platforms cover only some of the features of Multi-Agent Systems, like single agent platform, mobile agent platform or interaction infrastructures toolkit. In particular, few of them deliver a complete development environment supporting users in every development stage -analysis, design, development and deployment- for the creation of Multi-Agent Systems.

The work presented in this paper is part of a larger project aimed to develop a complete support for designing, developing and running MASs. The project purpose is to provide agent software developers with a language and needed infrastructures for Multi-Agent Systems development. The definition of the language, *L*MASS*, is in progress. This paper describes the framework that will support the execution of L*MASS. For the design of the framework, an object-oriented approach has been taken. The object-oriented paradigm is suitable for agent implementations since it models the world by independent entities, the *objects,* which communicate with each other by means of messages. Moreover, available object-oriented platforms allow objects to be autonomous, and to be distributed over a computer network. These properties facilitate the task of developing agent systems [16].

*L*MASS* agents are characterized by spatial position and internal state that determines whether and how they

react to perceptions. The *L*MASS* language has been defined according to an agent conceptual model named Reaction Diffusion Machine (RDM) model [13]. The RDM model allows for the simulation of complex systems in which entities react locally with each other and with the environment, and the global system behavior emerges from the local behavior of the composing entities. In the RDM the control is fully distributed. The agent behavior is determined by a local computation based on its position and sensitivity to fields as well as on reaction and diffusion patterns characterizing its type.

Agents defined according to *L*MASS* are based on a perception-deliberation-action mechanism, rather than on a simpler perception-action mechanism that impose for the agent an automatic and predefined response to every possible perception. Within this framework it is thus possible to describe agents characterized by a set of possible actions and a mechanism for the selection of the action to be undertaken based on the internal state of the agents. Agents described with *L*MASS* are *situated*, that is, their actions are also influenced by their position in the environment. The position in the space defines the situation in which the agent is acting, where 'situation' refers to a potentially complex combination of internal and external events and states. Reactive situated agents are very sensitive to the spatial relationships that determine constraints and capabilities for actions as well as privileged cooperation relationships. The space where agents are situated can reproduce a physical space, but it is nevertheless possible to define a "virtual space" that agents can roam and where they interact. Interactions take place when two or more agents are brought into a dynamic relationship through a set of reciprocal actions. Thus, interactions develop from a series of actions, whose consequences in turn have an influence on the future behavior of the agents [14].

The natural domain for *L*MASS* is the Multi-Agent Based Simulation (MABS). MABS is based on the idea that it is possible to represent a phenomenon as the result of the interactions of an assembly of agents with their own operational autonomy. Sometimes the Multi-Agent based approach to simulations is referred by the expression individual-centered simulation to highlight its aspect in studying and analyzing a complex problem as the effect of interactions of an assembly of simple and autonomous entities. Multi-Agent System can be used to simulate many types of artificial worlds as well as natural phenomena [14]. Agents could make the computer became a virtual laboratory where the researcher could modify experimental parameters to validate his model by observing and evaluating the system evolution.

Although models and systems to simulate physical phenomena originated the design of *L*MASS*, the approach is general and can be applied to the development of general-purpose MAS. A Multi-Agent System for the simulation of the chemical extraction of substances in washing phenomena occurring in percolation processes [15] has been implemented using *L*MASS* and it is currently in progress the design of a system for the simulation of localisation dynamics of shopping centres in extra-urban areas is in progress. Moreover, in order to validate the approach outside the MABS domain, it has already been applied to problems of awareness in CSCW applications [16].

The next section presents the elements of the *L*MASS* language. Section 3 discusses L*MASS action and interaction mechanisms to address the development of the framework for supporting the language. Section 4 discusses some implementation issues. Finally, section 5 concludes the papers with a description of the ongoing activities.

2 *L*MASS* elements

*L*MASS* is a language for the development of systems of situated agents. *L*MASS* defines a space, a set of *sites* whose interconnection is defined by an adjacency relation. The space of *L*MASS* is populated by a system of situated, autonomous and reactive agents that is characterized by dynamic evolution, and heterogeneity. Each site can host at most one agent. External *fields* propagating in the space influence the behavior of agents. Moreover, agents influence each other by perceiving presence and state of neighborhood.

Agents may execute specific actions (trigger, reaction, transport and field emission) as a consequence of interaction with other agents and of perception of fields. The possibility to specify different types of agents allows the definition of heterogeneous systems, where agents with different features and capabilities can coexist and interact.

In the following, each element of *L*MASS* is discussed to outline the peculiarities that the framework needs to address. For a more complete and detailed explanation of the current definition of *L*MASS* language see [20]).

2.1 Agents

The space of *L*MASS* is populated by a set of individuals called *agents* (Figure 1). A type defines each agent by describing the state of the agent, its interaction with the environment and the set of actions it can perform. Behavior and abilities of an agent define whether and how the agent changes its state and/or its position, how it interacts with other agents, and how neighboring agents can influence it.

According to the literature [3], *L*MASS* agents are entities capable of interacting with their surroundings (perceive the environment and act by executing actions), after *deliberating* what to do. Deliberation is affected by

the *perception* of the environment, i.e., fields that are present in the site where the agent is situated, presence of neighboring agents, and their state. Deliberation identifies which action, among a set of enabled actions, is to be executed by the agent.

Figure 1. An agent of type 'white' situated on site *p*. It can emit fields 1 and 2 and it is sensitive to fields 3 and 4.

The internal architecture of *L*MASS* agents is inspired by Brooks' *subsumption architecture* [19]. An agent is composed of three modules dedicated to each of the tasks the agent can perform (respectively perception, deliberation, and execution). Perception is a function that associates a set of influences with the set of possible actions. Execution corresponds to the application of an operator to produce an action based on the set of operations that can be carried out by the agent (according to its type, internal state and position). The phase between perception and execution is called deliberation, that is, the component of an agent responsible for its actual behavior.

The internal architecture of an agent is composed of an engine, a knowledge base and a rule base. The engine component implements the general behavior of any type of agent. As described above, every agent perceives, deliberates and executes. The knowledge base and the rule base define the particular set of information and actions a specific agent deals with. The knowledge base includes information on fields' values and status of neighboring sites and agents, over information on the agent itself (state, position, sensitivity to fields, and so on). The knowledge base is updated either automatically any time the environment changes, or by the agent itself as result of performed actions. The rules are defined by preconditions and actions to be performed. Deliberation consists in evaluating preconditions for rules activation and selection of the rule to be executed among those activated. Deliberation process can be customized to give tailored selection criteria. The default criterion is to select the rule that has been activated by the more recent change occurred in the agent knowledge base. This criterion has been chosen in order to ensure immediate reaction to changes in the surrounding environment.

2.2 The space

Agents are situated in environment that can be modeled as centralized or decentralized [14]. In the former case, the environment is described as a single block, while in the latter it is as composed of a network of cells. In centralized models agents can access a shared workspace that represents the whole environment and serves as interaction medium: an agent can leave messages for other agents and collect messages addressed to it. On the other hand, in decentralized environments each network node is a sort of centralized microenvironment that simulates a shared workspace to agents that exchange messages and signals as they were in a centralized environment.

The *L*MASS* space is a decentralized environment that is defined as a connected undirected graph, with an adjacency relation defined for the nodes (see Figure 2). Each node is a *site* hosting at most one agent. Fields propagate along the edges of the graph and leave on sites information about their presence. Sites are perceived as passive entities that store knowledge about the hosted agent, agents hosted by neighboring sites, and present fields (if any). Agents perceive the environment only locally through the sites that host them.

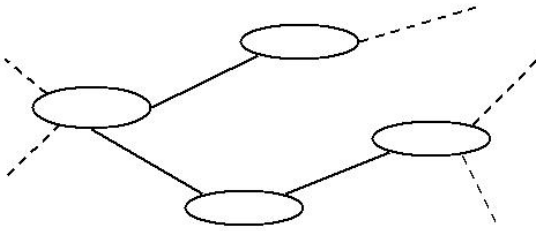

Figure 2. The L*MASS Space

2.3 Fields

Fields are the mean for agents to communicate asynchronously with each other and for the environment to affect the agents' behavior. Fields are generated by agents or by sources outside the space. In the latter case, fields model the interaction between environment and agents.

A field is characterized by a source value, a distribution function specifying how it propagates in the space, and by a signature allowing composing and comparing its values. Edges connecting sites define possible paths that fields can follow to propagate.

3 *L*MASS* Actions and Interactions

Agents can perform actions to change their state and position, and to interact synchronously and

asynchronously with other agents. Agents' activity deals with perception of the environment, reaction with neighboring agents, emission of fields, position change in the space, and state update. Activities of agents are defined by rules that take the form:

when *condition* do *action*

Condition is a Boolean expression that determines if the action may occur. The condition expression may refer to the internal state of the agent, values of the fields at the site hosting the agent, to neighboring sites and agents. Therefore, rules define three kinds of activities with respect to the interaction needed to evaluate the condition and subsequent action: Reaction that involves agent-to-agent interaction (a n-to-m relation); Transport that involves agent-to-site interaction (a one-to-one relation); and the reflexive interactions field emission and state updating.

In the following, the three kinds of activities are examined and discussed, along with the perception of the environment by agents.

3.1 Perception

An agent perceives the environment through the site that hosts it. These information are necessary to let the agent react to the environment and to make decisions on what to do. Three kinds of information are available: fields that are active on the site, the state of neighboring sites, and the state of neighboring agents.

When the state of an agent changes, or when an agent arrives at a site, the agent communicates its state to the hosting site that in turn forwards the information to the connected sites. In such a way, agents are constantly updated on the status of neighboring agents: whenever fresh information is needed, agents can ask the local site. In the same way sites communicate with connected sites about their status: free or occupied.

With respect to fields, two aspects need to be considered: field diffusion and field perception. Field diffusion specifies how the field propagates and composes its values. As already said, a field is originated by an agent in a site, and propagates over the space according to a propagation function. This means that propagating along a path, the intensity of the field change, and that agents perceive the field as it is locally. Moreover, since a field can propagate along different paths in the non-regular spatial structure, it is necessary to specify how the resulting different values have to be composed. Another aspect deals with the presence of more fields of the same type, even in this case field values need to be composed to define a single value at a specific site.

Field perception depends on the capabilities of agents. The agent type specifies sensitivity to fields defining thresholds that fix lower bounds to perceive fields. This is the very essence of the broadcast interaction pattern that underlies the field diffusion: fields propagate by means of messages that are not addressed to specific receivers but potentially to all agents populating the space. The set of potential receivers is determined first of all by the strength of the propagated messages, which can be null beyond a given distance from their source, according to the propagation function. Receivers of a field are free to decide whether and when deal with it.

3.2 Reaction

Reaction defines a synchronous interaction among a set of adjacent agents (that is, agents situated on adjacent sites connected by an edge). Agents react according to their type and state. Reaction takes place according to a state transition function and specifies how an agent can influence and be influenced by neighboring agents, that consequently might change their states. Reaction activity is defined by a set of rules as specified above.

The reaction process can be split in two phases: *negotiation* to agree on what to do and *execution* of what agreed. That is, the condition of a reaction rule needs to value true in order to execute the associated action. Since the evaluation of the condition involves information on the state of neighboring agents, synchronization mechanisms need to be introduced to ensure atomicity to the *negotiation* phase. In fact, during negotiation agents need to assume that the environment does not change; otherwise no decision can really be taken.

To avoid deadlock problems that may occur due to distributed synchronization, conflict resolution criteria need to be defined. It is well known that a general solution is unaffordable; therefore the *L*MASS* approach is to provide a default mechanism and let application designers to implement tailored criteria for the specific domain they deal with. The default mechanism is still under definition. The candidate proposal is to assign well-known priorities to agents to define who have the right of preempting others in case of conflict. Priorities should change dynamically to ensure fairness and avoid starvation problems. The effectiveness of any solution strongly depends on the nature of the application. For example, if few agents that seldom interact, or that interact only one-to-one, populate the space, it is likely that conflicts will not arise. Instead, in a crowded space, populated by agents that interact with every neighborhood, important conflicts will arise. In this case, the best solution would be to design ad-hoc criteria that take into account the nature of the application.

3.3 Fields emission and state updating

Actions execution by agents may change their internal state and cause the emission of a field. Field emission and

state updating are internal operations that anyway influence the environment. In fact, neighboring agents need to perceive these changes to define their behavior, i.e., activate reactions. *L*MASS* specifies that agents notify their hosting sites about state changes and field emission, so that they can propagate the information to connected sites.

Fields model asynchronous, long distance communication achieved by the propagation of fields in the space. In other words, these influences can be interpreted as broadcast messages that reach the various sites, possibly modulating their strength as specified by a propagation function associated with the field, and can be perceived by all the agents located in these sites, according to their sensitivity, type and state. The asynchronous interaction between agents is performed through the combination of fields emission defined as above with perception defining the effect of fields on agents.

3.4 Transport

Transport defines the rules that make an agent change its position in the space. In this case, the condition of a transport rule deals with the state of the neighboring sites, i.e., an agent may decide to move to empty sites only. Even in this case conflicts on occupying a site may arise. As for reaction, the approach is to provide a default criterion for conflict resolution, and leave the possibility of defining ad-hoc criteria when necessary.

The default mechanism is to serve the first agent that requests to move on a site, i.e., if an agent is willing to move to a site, it expresses this desire by asking the site status, consequently the site is considered locked if free, or unavailable if it is already occupied or it has already been contacted by another agent. This solution prevents from deadlock problems, but it may be inadequate for complex applications. In this case *L*MASS* allows designers to define more sophisticated policies. For example, an application may require a set of agents negotiate explicitly to designate who has the right of moving to a certain site.

4 Implementation issues

The implementation of *L*MASS* is to provide the language with a distributed environment to support the execution. Since the major objective of *L*MASS* is to define a simulation environment for complex systems, efficiency and performance are crucial properties. Nevertheless the development of a performing environment should not disregard the good software engineering principles.

An object-oriented approach has been chosen to keep the implementation of the language and its environment reflecting the *L*MASS* concepts even at run time. Moreover, Java has been chosen as development language since it provides a clean and comprehensive object-oriented development environment that covers distinctive aspects like distribution and code mobility. *L*MASS* defines the following entities: sites -to model the space-, agents -to model the active, mobile components-, and fields -to model asynchronous communication-. The *L*MASS* execution environment provides the language with support for configuration, deployment and execution of Multi-Agent Systems.

The system configuration is described by means of specific constructs that support the definition of the simulation space by interconnecting sites, the initialization of agents by creating and situating them, and the allocation of the fields on sites. In a simulation environment is important to separate the configuration and initialization phase from the actual system behavior for two reasons: usability and performance. In fact, configuration and initialization may change from time to time to experiment different solutions, and they should be executed only once without affecting the subsequent system evolution.

An important issue is the physical location of a site. Since *L*MASS* assumes a distributed environment, the run-time support has to provide for location transparency with respect to local or remote sites. In fact, any machine may host a site; designers of *L*MASS* systems should not be aware of that. At configuration time the environment has to set up the system according to the requirements of the current simulation run.

To support the execution of agents' activities, the run-time support needs to implement the interaction mechanism described in the previous section. Transport is the most critical activity, since it may require the move from one machine to another in a distributed environment. As for site location, the environment makes the distinction between local and remote move transparent to application designers. Field propagation is another critical issue. The environment has to ensure that every time there is a change in either the intensity or the location of a source, the effect of that change is perceived by the agents. Transport and reaction activities require synchronization. The run-time support makes synchronization transparent to agents by providing policies. As described above, policies are system-wide and can be changed by designers by replacing the default policies already embedded in the run-time support.

5 Conclusions and future work

*L*MASS* is ongoing projects that will deliver a language, and a framework that support its execution. This paper dealt with the definition of the framework that

supports the execution of the language. The development based on object-oriented paradigm should deliver a clean and flexible framework that can be easily tailored to accommodate several kinds of simulations to fit specific domain requirements. Literature reports successful stories about agent-based systems for scientific computing [18]. Anyway, the use of object-oriented platforms for scientific simulation is still questionable, since the costs of the infrastructure to support the execution of an object-oriented system could be too costly in term of performance. A future goal is to evaluate this aspect by benchmarking the parallel implementation and the distributed, object-oriented implementation of the tool.

The implementation is developed in Java to ensure portability and open the possibility of different execution patterns, ranging from concurrent execution on a single virtual machine to a truly distributed execution over networked computers. Preliminary versions of both single-processor and multi-processor implementations of the run-time environment for *L*MASS* are already available.

The future activity will deal with the completion of the definition of the framework, and the definition of the language constructs. Major issues that need to be addressed deal with a cleaner definition of fields and their propagation over the space, and the validation of the default synchronization mechanisms.

References

[1] Jennings, N. and Wooldridge, M. Intelligent agents: Theory and practice Knowledge Engineering Review, 10(2), 1995.

[2] Genesereth, M.R. and Nilsson, N.J. Logical Foundations of Artificial Intelligence. Morgan Kaufmann, 1987.

[3] Russel, S. and Norvig, P. Artificial Intelligence: a Modern Approach, Prentice Hall, NJ, 1995.

[4] Reticular Systems, Inc. AgentBuilder: an Integrated Tookit for Constructing Intelligent Software Agents, February 1999, http://www.agentbuilder.com.

[5] Busetta P., Rnnquist, R., Hodgson A., Lucas A., JACK Intelligent Agents -- Components for Intelligent Agents in Java, http://www.agent-software.com.au/, 1999.

[6] Bellifemine, F., Rimassa, G., Poggi, A., JADE - A FIPA-compliant Agent Framework. Proc. of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.

[7] Chauhan, D. and Baker, A., JAFMAS: A multiagent application development system. In Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), Minneapolis, ACM Press, May 1998.

[8] Gutknecht, O. and Ferber, J., Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Technical Report 97188, LIRMM, 161, rue Ada - Montpellier - France, dec 1997.

[9] Nwana, H. S., Ndumu, D. T., Lee, L. C. and Collis, J. C., "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems", Applied Artificial Intelligence Journal 13 (1/2), 129-185, 1999.

[10] Rao, A., Georgeff, M., "BDI agents: from theory to practice", Proceedings of the International Conference on Multi-Agent Systems, ICMAS-95, 1995.

[11] Finin, T., Fritzson, R., McKay, D. and McEntire, R., KQML as an agent communication language. Proc. of the 3rd International Conference on Information and Knowledge Management (CIKM'94), 1994.

[12] O'Brien, P. D., and Nicol, R., C. FIPA - towards a standard for software agents. BT Technology Journal, Vol.16, no.3, pp 51, July, 1998.

[13] Bandini, S. and Simone, C., Integrating Forms of Interaction in a Distributed Coordination Model, *Fundamentae Informaticae*, in press.

[14] Ferber, J. Multi-Agents Systems: An Introduction to Distributed Artificial Intelligence Addison-Wesley, Harlow (UK), 1999.

[15] Bandini, S., De Paoli, F., Manzoni, S., and Simone, C. OO Reactive Agents for RDM-Based Simulations. Proc. of AI*IA/TABOO Joint Workshop (WOA 2000), 2000.

[16] Bandini, S. and Simone, C. The reaction diffusion metaphor for modeling cooperative work. Prestige Journal of Management and Research, 1998.

[17] A. M. Uhrmacher, Concepts of Objects – and Agent Oriented Simulation, *Transactions on SCS*, Vol. 14. No. 2, 59-67, 1997.

[18] R. Gustavsson, Networked Agents for Scientific Computing, *Communication of the ACM*, Vol. 42, N. 3, March 1999.

[19] Brooks, R.A. A Robust Layered Control System for a Mobile Robot IEEE Journal of Robotics and Automation 2(1), 1986.

[20] Bandini, S., De Paoli, F., Manzoni, S., Pavesi, G., Simone, C. *L*MASS*: a Language for Situated Multi-Agent Systems, Internal Report University of Milano-Bicocca