A Generic Platform for a Multi-Agent Systems

Rachid El Bejjet, Hicham Medromi

***Abstract*— The main goal in this project is to define and implement an evolutionary and generic platform for multi agents systems (MAS) simulation by an agent-based approach. This paper describes an architecture conceived and developed in this area. Its principle consists of dividing and modelling a multi agent system to as many agents as necessary, an organisation and an environment. A designer or a programmer builds its application by making a choice of the necessary agents, by coding their processing around a processing model and by associating them around integration architecture, and then by coding non generic agents. This on agent based approach, conceived around a core to which we can add and develop thematic software layers, enhance the modularity and the genericity of developments allowing their reuse in future developments. This characteristic distinguishes our platform from other existing environments for being not only independent from any other particular multi-agents model but it offers a generic models' library as well.**

Index terms— ***Multi agents systems, simulation, distributed intelligent platform, integration architecture.***

I. INTRODUCTION

Currently the computer systems are more and more complex, often distributed on several sites and constituted of software in interaction between them or with human beings. The need to use the technology agents become obvious and the evolutions in this area are remarkable. In this context, the conception and the development of a multi-agents system (SMA) are complex problems because they require taking into account several parts of the system that can often be tackled under different angles. The designer must first identify the set of the problems then to solve, and then find some multi-agents models for their resolution implement them and then integrate them in a coherent system.

Manuscript received June 15, 2010.

R. El Bejjet is an engineer and as a PhD student, he joined in 2008 the system architecture team of the ENSEM, Casablanca, BP 8118, Morocco. His actual main research interests concerns on agent based computing. (tel: +212661709287; fax: +212522231299, e-mail: rachid.elbejjet@gmail.com).

H. Medromi received the PhD in engineering science from the Sophia Antipolis University in 1996, Nice, France. He is responsible of the system architecture team of the ENSEM Hassan II University, Casablanca, BP 8118, Morocco. His actual main research interest is Control Architecture of Mobile Systems Based on Multi Agents Systems. Since 2003 he is a full professor for automatic productic and computer sciences at the ENSEM, Hassan II University, Casablanca, Morocco. (e-mail: hmedromi@yahoo.fr).

These tasks justify the use of development environments that helps the designer by providing him with tools and models already developed on which he can rely.

There are a number of multi-agents models that can be independent, competitive, complementary or incompatible. The combination of several models to construct a complex application is a non trivial problem. Most environments exist Jade [3], FIPA [8], Madkit [9], Zeus [10] are founded on a main model, what avoids the problem of consistency between models but restricted the categories of applications to those targeted by the model. Some recent environments such VOLCANO [11] introduced the idea to use a model of component to express the dependences between models and to combine them.

This paper proposes a generic platform used in a distributed environment. It has been conceived for the simulation of the multi-agents systems, and based - itself – on the agents approach, and by the way answers to needs, in trials, development and implementation of the MAS applications.

We will show how this architecture allows us to implement different execution policies able to coexist inside one only and same simulation.

The on-agents based approach adopted is presented then in the section 2, the platform proposed for the simulation of the multi-agents systems is discussed in section 3. The section 4 exposes an illustrative example of application by giving a process of an application's simulation achieved through our platform.

II. ON AGENT-BASED APPROACH

In this part, we will give some basic and well-known architecture's model to illustrate our approach in terms of agents and multi agents systems.

A. Basic model of a situated agent

For a certain proportion of multi-agents simulations (and also for other types of applications, for example in robotics), the agents are situated in an environment, that they discern through their sensors, and on which they can act and can produce some effects. The calculation's cycle (at every step of simulation) is usually the next one: the agent discerns via his sensors its environment (in particular, the presence of other agents close by, of obstacles, of pheromones. . .); these data

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

are processed by its behaviour (intern); to produce, via its effectors, actions on the environment (ex. moving, taken of ball or object, deposit of pheromone. . .). The general model of a situated agent follows the cycle as follows:

sensors → behaviours → effectors

Therefore, the abstract architecture model in figure 1.

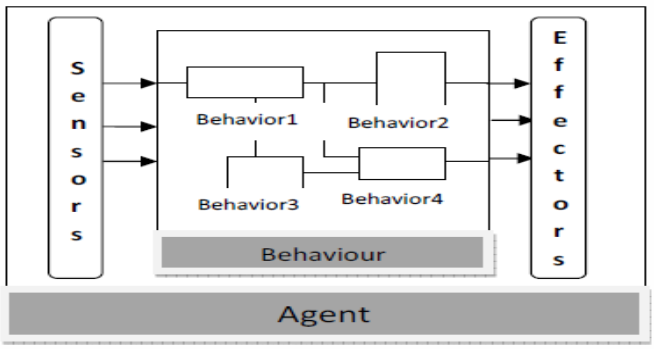

Figure 1. General architecture of a situated agent [18]

B. Multi agents systems

According to the model **Vowels** of Yves Demazeau, a MAS can be defined as:

- a set B of entities put in an environment **E** (E is characterized by the set of the states of the environment S)
- a set A of agents A⊆B
- an action system operations) allowing these agents to act in E (an operation is a function of S => S
- a communication system between agents (mails sending, signal broadcast, … (**I** as interaction)
- an organisation **O** structuring the set of agents and defining the functions in charge of agents (notion of role and eventually of groups)
- eventually: a links to users **U** who act in this MAS through agents interfaces U⊆A

Furthermore, we consider three levels of analyse witch are: the agent, the interaction and the organisation.

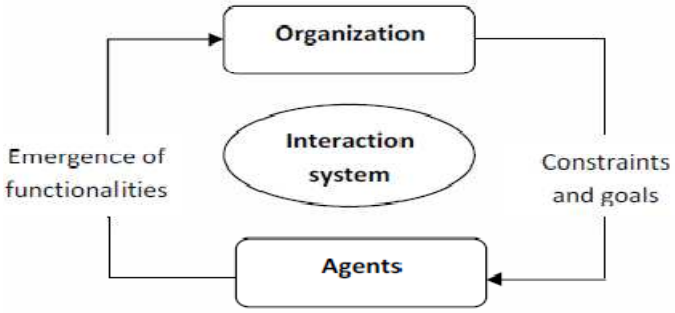

Figure 2. Analysis levels of a MAS application [18]

III. DESCRIPTION OF THE PROPOSED PLATFORM

The platform should not be limited to a type of specific MAS and should remain generic and expandable, so we chose a modular approach: The core only structures the application and these are the modules that provide some functions to the application. So, for a particular application, only the necessary modules are invoked and are used.

In this way the MASSDIP platform (Figure 3.), remains the fastest and lightest possible by avoiding any useless code charged in memory. At the same time it can adapt to different types of MAS thanks to the addition of suitable agents. In counter part, being intended to a "professional" use, the platform requires for its use, some skills in Java programming.

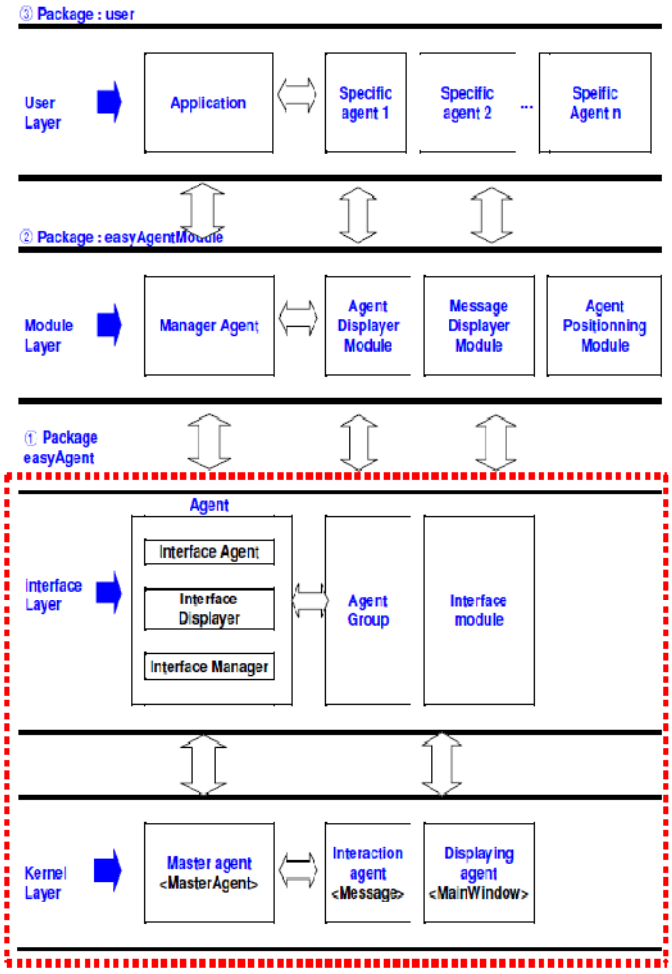

Figure 3. A functional architecture of the platform MASSDIP

The platform is conceived in three layers, each can be modelled by a package: A central part (package easyAgent), a modules part (package easyAgentModule) and a user part. It is to note that the central part can be compiled regardless of the

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

modules part and the user part. In the same way the modules part can be compiled without the user part.

A. The central part

The central part is designed in two packages: kernel and agent interfaces. The package kernel is intended to integrate the system code of the platform. This package is in charge of the management of the modules and agents. The package agent contains all interfaces and abstract classes that it is necessary to redefine in the modules and the part application.
The package kernel consists of three important classes:

- Master agent (Masteragent),
- Interactions agent (Message),
- Display agent (MainWindow).

- **MasterAgent** is the main agent of the platform that manages all the application. It is the class that manages all the application. MasterAgent is a singleton and we can recover its process thanks to the function getInstance (). MasterAgent contains several functions allowing to start the simulation, to stop it and to leave the program that is respectively start(), stop() and quit(). MasterAgent possesses an IAgentManager by default that it is possible to inform by the setDefaultAgentManager() function and to recover by the getDefaultAgentManager() function.
MasterAgent forks also a window that it is possible to recover by the function getMainWindow(). The modules of display are added automatically to the main window by the function addDisplayer(IDisplayer) of MasterAgent.
- **MainWindow** is the main interface of the program. It places all IDisplayer on a GridBagLayout. The window manages some Components. It lets some freedom to the programmers to implement the displays of the modules. The Components are placed in a gridBagLayout. All IDisplayers must provide therefore the window with one Component and one GridBagConstraint.

- **Message** is the class that acts as envelope and post station for the classes coming from IMessageBody. The constructor of Message takes three arguments:
  - One IAgent witch is the mail sender.
  - One IAgent witch is the recipient.
  - One IMessageBody witch is the mail body

The package agent contains all interfaces that must be implemented by the different modules and agents. We will can keep the few important interfaces:

- Agent interface (Iagent),
- Display interface (IDisplayer),
- Manager interface (IManager).

We will note that the package agent contains a Group class that implements IAgentContener. The Group class records the agents that he contains in a TreeMap. In the same way the package contains a class Agent that records itself automatically by the defaultAgentManager.

B. The modules agents

If the core ensures organizing and structuring the application, the modules provide the means to achieve it. It requires the setting up of specialized agents, regrouping some interdependent classes, these classes should implement or extend the interfaces or the classes defined in the easyAgent.agent packages or easyAgent.core.message. The classes can be mainly managers, displayers or define bodies of specific message to the module.
The package easyAgent.agent.manager has two interfaces :

- IAgentManager
- IspaceManager

These interfaces that inherit IAgent and IAgentContener and must possess a means therefore to contain the agents that record themselves. The definite classes should, of course, define all abstracted functions of these interfaces, but it is to note that a Group class IAgent and IAgentContener have already been defined in this package, defining all these functions of the classic manner of use and using a TreeMap in which it arranges the agents that are recorded themselves, one can recover these agents then from their name, or in the alphabetic order of this one. We can extend Group therefore and can only redefine the specific functions to the module.
The displayers must implement interfaces IDisplayer of the easyAgent.agent.displayer package, if they must only display an agent, they can implement IAgentDisplayer. They must define getComponent() and getGridBagConstraints() that return the element to display to the screen as well as the constraints to place it in the GridBagLayout of the main window. The body of a message should implement IMessageBody of the easyAgent.agent.message package and will be able to contain some attributes or no.

- *Manager module*

This agent consists of an AgentManager class that implement IAgentManager and extends the Group class. It contains a function run() that calls the function run() of each of its agents, in the order of its TreeMap. If an agent returns false, AgentManager is interrupted by itself and returns false. A process of this class is defined by the application as DefaultAgentManager by the MasterAgent.

- *Message Displayer Agent*

This agent consists of an unique AgentDisplayer class that implements IAgentDisplayer and extends javax.swing.JTextArea. It is used to display an agent's information in the main display window.

- *Message Displayer Module*

This module contains two classes, Message and MessageDisplayer.
The class Message extends easyAgent.core.message.Message and replace it in the application so that any sent message is displayed thanks to the MessageDisplayer.
MessageDisplayer implements IDisplayer and displays the messages in a textArea. The size of this one being limited, the display is interrupted by itself but the application goes ahead.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- *Positionning module agent*

This agent consists of three classes :
- SpaceManager2D: knows the place of any agent so to allow its display.
- SpaceDisplayer2D: displays the agents in their grid.
- IMove: body of message that agent should send to SpaceManager2D when moving.

IV. Experiment

The aim of this project is the conception and the achieving of a platform for multi-agents systems' simulation. It is proposed, here, the description of a simulation process for a particular on-agents based application.

A. Description of the application to be simulated

The example is about a simulation of a multi-agents system of a soccer team. (Figure. 4), designed as follows:

- Six cognitive agents: Who play the role of players, these players are cognitive by looking to have enough information on their environment (here the land) and to act according to the different situations. So each player is able to see both the ball and the other players, to know his adversaries and to react while taking in consideration all these information. Also, as in a team of football, here the coordination and the communication are indispensable for a good progress of all operation.

- Two reactive agents: These agents will be the goal keepers, the interest that these are reactive agents relies on the fact that a goal keeper doesn't react before the ball is close to the goal. Therefore an agent that plays a referee's role will react (to get ready to catch the ball) in the case where a ball will enter in a given perimeter.

- An arbitrate agent: This agent supervises the behaviours of the other agents of the two teams in order to not to pass the sides of the land. He is a reactive agent because he only reacts on the event that is the exit of an agent out of the land.

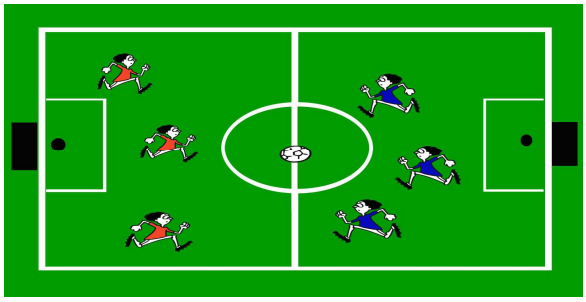

Figure 4. Example of MAS application to be simulated

B. Description of the simulation process

The tools chosen to develop this platform are *java, html* and associated tools. The choice of *java* is based on its well-known portability and web technology suitability.

Figure 5. User interface

Once the designer creates the agents and objects to the number and type required to its application, he has available means to create the rules of behaviours and models of knowledge for each of the agents. Finally, he can implement the logic of interactions by messages. The designer can start the simulation and can refine his application thus by multiplying the scenarios of its implementation.

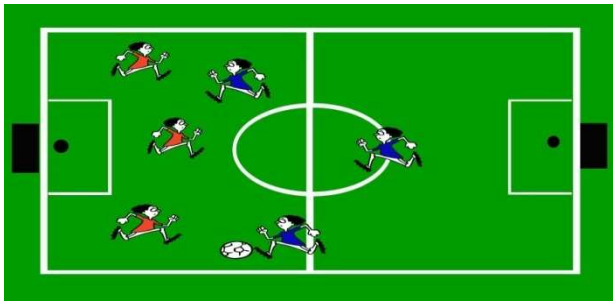

Figure 6. Simulation scenario of a soccer team

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

V. Conclusion

In this paper, we presented a distributed and intelligent platform intended to the simulation of the multi agents systems. This architecture is relatively original, because it is based on the agent's behaviour and also by the fact that it is developed directly by the Java language.

These features offer interesting possibilities of genericity and combination. Besides, it is simple of use, distributed and extendable, it provides as well as a precise control of the internal organization to an agent, witch is much valuable in particular for the of multi-agent applications' simulation field.

We think that there is no optimal architecture of agents; it depends notably on the area of application considered and on the needs. A general architecture (hybrid), as InteRRaP, that tempts to reconcile and factorise at a time the cognitive architectures and the reactive architectures, is powerful but also complex. To the contrary, our model of architecture is simpler; it limits itself in fact to a model by layers, the efficient architecture remaining to be define by the designer.

Our platform has been tested more especially on the applications of simulations developed within the EAS team of the ENSEM.

We hope to have been able to show some of its abilities. Its applicability to other types of trials and agents remained to experiment and to value. But the MASSDIP framework is in fact generic enough, and it is especially about having a library of agents and abstracted architectures appropriated to the types of applications and architectures considered.

References

[1] Drogoul A., De la simulation multi-agent à la résolution collective de problèmes, Thèse de doctorat, Université Paris 6, 1993.

[2] Guessoum Z., Un environnement opérationnel de conception et de réalisation de systèmes multi-agents, Thèse de doctorat, Université Paris 6, mai 1996.

[3] Briot J.-P., Demazeau Y. (ed), Principes et architecture des systèmes multi-agents, Collection IC2, Hermès, 2001.

[4] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., « Jade Programmer's Guide », JADE 2.5, 2002.

[5] Carabelea C., Boissier O., Florea A., «Autonomie dans les systèmes multiagents », JFSMA '03, 2003.

[6] Meyer D., Buchta C., «SIMENV: A Dynamic Simulation Environment for Heterogeneous Agents», Working Paper SFB, Nr. 100, August 2003.

[7] « Enterprise JavaBeans », http://java.sun.com/products/ejb/index.jsp

[8] «Foundation for Intelligent Physical Agents», http://www.fipa.org.

[9] GUTKNECHT O., FERBER J., MICHEL F., «MadKit: Une expérience d'architecture de plate-forme multi-agent générique», PESTY S., SAYETTAT C., Eds., JFIADSMA'00, St Jean-la-Vêtre, Loire, France, Octobre 2000, Hermès, p. 223-236.

[10] NWANA H., NDUMU D., LEE L., COLLIS J., « Zeus ; A Tool-Kit for Building Distributed Multi-Agent Systems», Applied Artificial Intelligence Journal, vol. 13, no 1, 1999, p. 129-186.

[11] RICORDEL P.-M., DEMAZEAU Y., «La plate-forme VOLCANO : modularité et réutilisabilité pour les systèmes multi-agents», Numéro spécial sur les plates-formes de développement SMA. Revue Technique et Science Informatiques (TSI), 2002.

[12] GUESSOUM Z., BRIOT J.-P., «From Active Objects to Autonomous Agents », IEEE Concurrency 7,(3), , 1999, p. 68-76.

[13] GUESSOUM Z., OCCELLO M., «Environnements de Développement», BRIOT J.- P., DEMAZEAU Y., Eds., Principes et architectures des systèmes multi-agents, p. 177-206, Hermès Sciences Publications, Paris, France, Décembre 2001.

[14] HUBNER J. F., SICHMAN J. S., BOISSIER O., «Spécification structurelle, fonctionnelle et déontique d'organisations dans les Systèmes Multi-Agents», MATHIEU P., MULLER J.-P., Eds., JFIADSMA'02, Lille, France, 2002, p. 205-216.

[15] G. Cabri, L. Leonardi, F. Zambonelli "BRAIN: a Framework for Flexible Rolebased Interactions in Multi-agent Systems", Proceedings of CoopIS 2003, 2003.

[16] E. A. Kendall, "Role Modelling for Agent Systems Analysis, Design and Implementation", IEEE Concurrency, 8(2): 34-41, April-June 2000.

[18] J. Ferber: Les systèmes multi-agents, vers une intelligence collective, Paris, InterEditions, 1995.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)