# GENA: MULTIPLATFORM GENERIC AGENTS

LAURENT MAGNIN[†] AND EL HACHEMI ALIKACEM
*Centre de recherche informatique de Montréal,*
*550 Sherbrooke Street West, Suite 100, Montréal, Québec H3A 1B9, Canada*
*E-mail: {lmagnin, alikacem}@crim.ca*

[†]*Also adjunct professor at the Université du Québec à Montréal*

Mobile agents in the Internet environment bring new constraints. In particular, various agent platforms will coexist on the Net. Our generic agent model "GenA" allows agents to run on and move between different agent platforms by using an interface called "GenAi." Programming these GenA agents is independent of the underlying agent platforms. An implementation of this agent model has been performed and validated by a small application.

## 1 Introduction

### *1.1 Context*

Distributed Artificial Intelligence (DAI) and Multiagent Systems (MAS) have contributed significantly to problem solving [3] and simulation research fields [2]. However, the Internet will be the next focus and application domain for DAI and MAS especially for E-Commerce through the use of "Intelligent Agents" [11; 15].

Nowadays, the Internet is mostly used as a set of channels through which e-mails and HTML pages are transmitted as requested by users. Due to its success, the Internet is overloaded with a huge amount of available and transmitted data. Help came firstly through the introduction of Search Engines. However, these are not fully satisfactory: (a) the research has to be done by the user, (b) using key words allows only basic research, and (c) the number of responses becomes more and more unmanageable. A more "intelligent" and "pro-active" Network is therefore desirable, one which will really interact with users by providing them with more accurate information even before they need to ask for it. To obtain such features, agent technologies seem to be an answer [5; 9].

### *1.2 Heterogeneous Multiagent Platforms*

Nevertheless traps to Internet agents are numerous from a research and a technical point of view as well as from a financing one. As a result, there is currently no significant application which would be based on Agents and designed for large Internet audiences [7; 8]. It is expectable that most of the companies will only show some interest once the technology is implemented.

Today, we sit on the edge with, on one side, huge promises (consider the tremendous interest existing on the stock market for E-business companies) and on the other side many difficulties. These obstacles are both commercial (many companies have to be involved in such projects to provide useful solutions) and technical, and arise from the following:

- Agent's intelligence, e.g. its ability to cooperate.
- Mobility and moving constraints.
- Design and implementation of multiagent methodologies, *i.e.* Agent Case Tools.
- Need for generic agents able to cope with various protocols and commercial multiagent platforms evolving on the Internet.

Only the last aforementioned point concerning generic agents will be covered in this paper. Indeed, one has to realize that the Internet creates a lot of new constraints most of which have never been experienced before. One of the major constraints for the to be agents resides in the ability to adapt to an odd environment (co-existence of various platforms), pre-existing to them (with other agents having their own specific background) and alien (*i.e.* produced by other companies which may even not exist yet). This is contrary to the traditional MAS in which the designer of this particular MAS produces all its agents. As a result most of the time no distinction is made between the agent pattern and the basics allowing it to run (*i.e.* its agent platform).

Let's illustrate this situation with the following application dealing with a mobile agent [14] whose purpose is to find the cheapest flight ticket on the Internet [10]. How to manage the case where the Air Canada server is designed for IBM Aglets agents [1] whereas TWA works with Voyager agents [13] from ObjectSpace? Would the designer of the mobile agent be obliged to write two specific applications? How to face a third Airline Company whose platform is different, and ultimately unknown?

### *1.3 Fit for purpose solution: GenAi interface*

One of the immediate answers for solving the problem deriving from incompatible multiagent platforms would consist in implementing standards [4; 12] that need to be massively used. Unfortunately, this option seems to us unrealistic, at least in short time, due to commercial and technical reasons. The commercial reasons lie in the fact that although many companies want to impose their own standard, no one owns the power do so, and they all know that adopting a solution provided by another company will mean losing the war. The technical reason is that founding an universal agent model is not an easy task, notably more difficult than adopting a "classical" object model (based on attributes, methods, hierarchy of classes, etc). In fact, developing a new multiagent standard is similar to developing a complete new operating system, and very different from developing, for example, HTML or XML that could be described by grammars.

Another solution could be to launch a universal platform complying with others. But this would imply to set it up on all agent servers existing on the Internet. This suggestion is as much challenging as the previous one.

Finally, the third way would be to define universal agents compatible with any platform. This option does not appear to be technically realistic, based on our review of the current platforms available on the market. Indeed, it would involve coping with multiple inheritances without knowing application source codes.

Our approach is different: it consists of developing an intermediate layer called "GenAi" between our "GenA" agents and the targeted agent platforms (all Java based). In that way our agent will be able to run and eventually talk to native agents on these different agent platforms whilst keeping the same functionality, thanks to interfaces. That does not imply that all of the platforms need to integrate these GenAi interfaces: it is only when a GenA agent reaches an agent server that the GenAi Java classes need to be downloaded by the Java Virtual Machine (JVM), without modifying the server behavior. This way, we don't impose a new standard, or more precisely people who want to use the genericity of GenA agents don't need that GenAi becomes a standard.

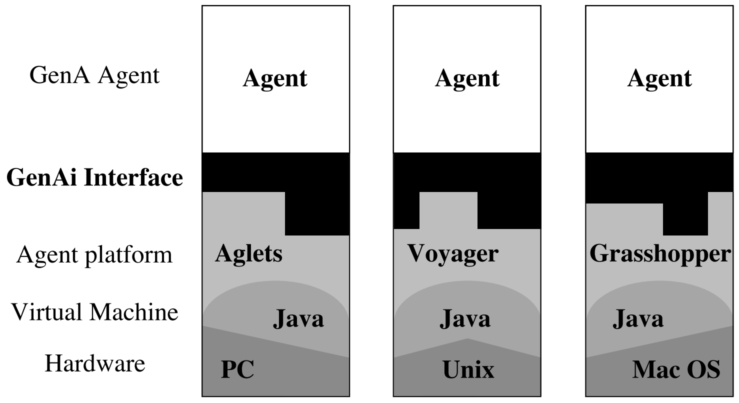

**Figure 1.** GenAi interface

## 2 Generic agents

### *2.1 Introduction*

Usual agents are "specific" to a platform, *i.e.* they are designed for this platform and can not run on another. Adversely, a "generic" agent is not dedicated to a unique platform. This definition doesn't mean that this generic agent is abstract without relation to any platform. For us, an agent is "generic" when it is able to run on different agent platforms.

Once definitions clarified, we intend to demonstrate in that paper the difficulties inherent to such generic agents and the answers to overcoming them. Our approach is pragmatic based: instead of developing a universal model that would then require to be fitted to the targeted platforms, the approach is incremental and experimental. Three multiagent platforms have been selected for which our GenAi interface has been developed step by step (creation, execution loop, migration, etc.). Each of these functionalities will be introduced below, with due consideration to the difficulties they imply and to our approach to sort them out.

## *2.2 Generic agent / native agent*

### 2.2.1 General comments

A specific agent is made up of two facets, one is specific to the platform expected to carry the agent, the other one is independent from any platform. Moreover, in order for this agent to move from one platform to another, it should be able to change dynamically its specific facet whilst maintaining its internal status.

It is generally impossible to get access to the source code of agents associated to the targeted platforms. As a result, it is impossible to implement these generic agents by a modification of their code which is offered by the platforms. We must be satisfied with the public methods proposed by these agents. Therefore, our solution consists in designing a GenA agent modeled as the sum of two interconnected objects. The first one (so called “native agent”) inherits from the agent class of the targeted agent platform. The second object (so called “generic agent” with the purpose of implementing the agent function) inherits from the generic agent class GenA. Consequently, our agents are simultaneously perceived by platforms as being native and by their originators as being non specific GenA agents.

A critical constraint has to be mentioned: all the selected platforms require using their own Java Virtual Machines. When a composite GenA agent is created, the targeted JVM has to be able to fully integrate this agent whereas its creation has been initiated in another JVM. For doing so, the generic agent has to be created on the original JVM and then handed over to the targeted JVM as a copy at the time the native agent is created. One of the conditions for the agent migration is to ensure that the GenA agent is serializable.

### 2.2.2 GenA & GenAi implementation

As aforesaid our GenA agents must have GenAi interfaces adapted to specific multiagent platforms if they are intended to be operational. In the first stage of our project three Java based platforms have been selected:

- ASDK, Aglets Software Development Kit [1], developed by the IBM Tokyo Research Laboratory.
- Voyager [13], commercial product of ObjectSpace.
- Grasshopper [6], commercial product of German firm IKV++.

Three GenAi interfaces have been allotted to theses platforms, once the latter have been fully analyzed. For each platform, there is a server whose main purpose is to host agents by granting them some CPU time and basic services. Consequently, for each targeted host, one or several servers must be launched manually or automatically.

Any GenA agent is defined by a sub-class called “AgentGenA”. Two main methods have to be defined in this class:

- The “void behavior()” method: to define the agent behavior.
- The “ Object handle_GenMessage(GenMessage genMsg)”: to handle the messages received by the agent.

## *2.3 Execution Loop*

Implementing an agent requires offering to it regular access to CPU time (to say, allowing it to get “executed”). Most of the platforms ask for a specific agent method which is then called back by these platforms when executing the agents.

A very simple and fit for purpose solution has been found: the native agent method is forwarded to the corresponding methods of the generic agent (the “behavior” method as defined at the GenA agent class level).

## *2.4 Split up*

The purpose of our research is to ensure that agents can adapt to the Internet. This requires that these agents be able to run on remote computers. It is therefore necessary, besides the issue of their mobility, to ensure that these agents are created for remote computers. This raises the question of dynamic download of GenA & GenAi classes by the hosting JVM. Our experiences led us to the conclusion that there is no universal answer: in fact, it depends on the specific services provided by the multiagent platforms when native agents are created. In addition, the notion of proxy has to be introduced in order to maintain a link with the generated agents.

The instantiation of GenA agents goes through the creation of a proxy belonging to the class proxyGenA. The constructor method of this class creates the objects necessary to the agent, notably by calling the creation method of the native agent. The proxy remains on the virtual machine on which the creation code was done. Adversely, the agent will be located on the JVM of the targeted multiagent platform.

```
ProxyGenA proxyGenA =
new ProxyGenA("MyGenAAgent", "AG::/bilbo.crim.ca:8001")
```

This constructor is made up of two arguments:

- the class defining the GenA agent;
- the server address of the platform dedicated to the agent as well as the code specifying the type of platform (in that particular case “AG for Aglets”).

## *2.5 Mobility*

The simplest mobility for an agent consists in moving from one agent platform to another one of the same profile. Since all the platforms selected for our GenA agent own this migration functionality, the GenAi interface makes use of it. With one restriction: most of the time, the agent has to be asleep before being able to migrate. Its wake-up on the new platform is triggered by a specific method call off. Indeed, according to the terminology used by [10], the mobility level of the agent is "1": only data and procedures are conveyed to the exclusion of the status vector.

The main difficulty, and likely the beauty, of our approach is to allow GenA agents to move from one platform type to another. For doing so, particular mechanisms have been set up to make the generic part of the agent (and only it) migrate to the target platform, which will allot to it a new native agent. Nevertheless, the agent is the one which asks for self-recreation in an other place before committing suicide (whilst leaving a proxy so that one can maintain a link with it).

## *2.6 Communications*

Two communication modes can be foreseen for generic agents; either by using *ad hoc* codes, or by referring to the mechanisms of the related platforms. According to our project terms of reference, initial focus has been given to the latest communication mode. Indeed, this is the only one that grants some form of communication with the non GenA agents evolving on the platforms. The first step resides in harmonizing the way messages are shaped: they will appear as "GenMessage" serializable objects by definition accepted by the three selected platforms. Such a message sent by its agent is then transmitted to its associated native agent thanks to the GenAi lay. The native agent conveys this message to the native agent associated with the recipient agent. Then, the message is forwarded to the generic agent which actually interprets it through an *ad hoc* method. Communication between two different platforms is made possible through specialized coupled GenA agents posted on both platforms.

In order for a GenA agent to receive messages, the following method, which will be executed at the reception of the message, has to be defined:

```
public Object handle_GenMessage(GenMessage Genmsg);
```

Synchronic messages are sent to an agent by using the "send_GenMessage(GenMessage msg)" method of ProxyGenA class. This method refers to an object from the type "GenaMessage", which bears the name of the message and eventually a table of arguments. This method returns an object from the recipient to the sender.

Asynchronous messages (of "GenMessage" type) are sent to an agent by using the "send_oneWay(GenMessage Genmsg)" method of ProxyGenA class. The message sender is not blocked and does not receive any answer from the recipient.

## 3 Outlooks

The GenA project has started recently. Consequently, only the GenAi interface has been finalized so far. In order to validate our generic agent approach and the implementation of this interface, a demo application based on mobile agents GenA is being set up at Crim.

Some work still needs to be done for this GenAi interface. We need to:

- Increase the number of services offered (agent access through their name, communication through broadcasts, etc.).
- Strengthen choices made (parameters of methods calls, etc.).
- Implement new GenAi interfaces dedicated to newly set up multiagent platforms.
- Perform new experimentations, in particular about performance.

The rest of the GenA project will have the same purpose as the one developed in the GenAi interface, *i.e.* give generic agents (potentially described by meta-models) access to a maximum number of resources offered by applications developed by third parties. The main difficulty of this work arises from three directions. First, since these resources are on an open Internet network, they cannot be known from start. Second, they change based on the time and the server on which the agent is. Third, they don't give an access to their source code or to a detailed description of their services.

## 4 Conclusion

The implementation of agents in open and commercial environments such as the Internet, includes new constraints. Agents need to become generic, *i.e.* not linked to specific execution platforms. That way they will be able to run on a large range of agent platforms and *de facto* on most of the agent based servers.

It is why we propose in this paper a new model of generic agents that run on different agent platforms by using an interface called GenAi. Such GenA agents can move from one platform to another that are basically incompatible. Of course the communication between our GenA agents and native ones is also allowed.

Much research work still needs to be done in the generic agents field, notably by the way of a meta-modelisation of them. Nevertheless, thanks to the implementation of the GenAi interface, we have proved that such agents are technically possible, at least with a limited framework. Moreover, this implementation will enable further experimentation, potentially in real life.

## 5 Acknowledgements

We specially thank the BUL (Bell University Laboratory) which has, through the RCM2 / NCM2 (Network for Computing and Mathematical Modeling), supported most of the research regarding the GenAi interface implementation.

We acknowledge also the efforts of Didier Naranin and Nicolas Besson, students at the University of Paris VI (France), for building the GenA demonstration and validation application.

## 6 References

1. "Aglets Workbench by IBM Tokyo RL", http://www.trl.ibm.co.jp/aglets/
2. Magnin Laurent, "SIEME: an Interactions Based Simulation Model", Proceedings of ESM '98, Manchester, U.K., June 98.
3. Durfee, E. H., "Distributed Problem Solving and Planning", Multiagent Systems (ed. by G. Weiss, Cambridge, London, The MIT Press, 1999) pp. 121-164.
4. Fipa, "Foundation for Intelligent Physical Agents", http://www.fipa.org/
5. Gasser, L. and J.-P. Briot. "Agents an Concurrent Objects: Briot-Gasser Interview" http://www.lis.uiuc.edu/~gasser/AgentsAndObjects-07.html
6. Grasshopper, "Grasshopper by IKV++", http://www.ikv.de/products/ grasshopper/index.html
7. Joshi, A. and M. P. Singh (1999). "Multiagent Systems on the Net". *Communications of the ACM* **42** (March 1999) pp. 39-49.
8. Ma, M. "Agents in E-commerce", *Communications of the ACM* **42** (March 1999) pp. 79-80.
9. Maes, P., R. H. Guttman, et al., "Agents That Buy and Sell", *Communications of the ACM* **42** (March 1999) pp. 81-91.
10. Merlat, W., "Objets et agents pour Systèmes d'Information et de Simulation", Ph.D. thesis, University Paris VI, Paris, 1998.
11. Müller, J. P. and M. Pischel, "Doing Business in the information Marketplace", Proceedings of Autonomous Agents, Seattle, ACM, May 1999.
12. OMG, "MASIF: Mobile Agent System Interoperability Facility", http://www.fokus.gmd.de/research/cc/ima/masif/index.html
13. Voyager, "Voyager by ObjectSpace", http://www.objectspace.com/voyager/
14. Wong, D., N. Paciorek, et al., "Java-based Mobile Agents", *Communications of the ACM* **42** (March 1999) pp. 92-102.
15. Wooldridge, M., "Intelligent Agents", Multiagent Systems (ed. by G. Weiss, Cambridge, London, The MIT Press, 1999) pp. 27-78.