Agents Need to Become Welcome

Laurent Magnin[1], Hicham Snoussi[1], Viet Thang Pham[1],
Arnaud Dury[1] and Jian-Yun Nie[2]

[1] CRIM, Montreal, 550 Sherbrooke Street West, Suite 100
Montréal, Québec, Canada H3A 1B9
{lmagnin, hsnoussi, tvpham, adury}@crim.ca
http://www.crim.ca
[2] University of Montreal, Office 2227, André Aisenstadt,
CP 6128 succ Centre-Ville,
Montréal, Québec, Canada QC H3C 3J7
nie@IRO.UMontreal.CA

Abstract. To succeed, agents need to become part of legacy and future systems that are not agent oriented. That means agents must be able to run (and migrate) on servers that are not based on multi-agent platforms. Also, agents will have to be able to communicate using languages and protocols that are not dedicated to them, such as HTML. Last, agents need to be able to adapt (themselves) to modifications of their environment. Such features are provided by a family of agents developed at CRIM, which we call *Guest.*

1 Introduction

Software Agents have existed for about twenty years but their integration into real systems remains an issue. This is probably because we all, as researchers, want to develop new systems based on what we are working on, agents. However, even if such approach succeeds in limited or specific applications, where the architecture is based on multi-agent systems, it is quite difficult to "impose" our technologies to other field such as the Internet. So, we have to think about how to adapt agents to different domains instead of the opposite.

Since agents need basic facilities like recurrent access to the CPU, communication facilities, etc, they cannot run without *ad hoc* servers. Therefore, all agents must be linked to specific multi-agent platforms, which provide the basis of such servers. This means that if you want to use agents somewhere, you first have to start a multi-agent server there. For agent centric applications, this is usually not a problem. However, adding a new kind of applications like multi-agent servers to legacy general-purpose systems could be a challenge, not only because of technical difficulties but also because of fear of the unknown. To resolve such difficulties, agents will have to run on servers that are not dedicated to them.

Because communication languages for agents (ACL, KQML, etc) are too specialized and domain-specific, a "pure agent" cannot get and send information to legacy systems. Since it is impossible to add agent-based communication channels to all of such systems, we have to integrate general purpose (XML, RMI, ODBC, etc.) communication features directly to agents. In fact, because an agent is proactive, it can become a bridge between heterogeneous systems or services using different languages or protocols as used by the "software" it is speaking to, including other agents.

Closed and non evolving or short lived multi-agent systems (MAS) do not need maintenance capabilities, unlike agents running inside open and dynamic systems like the Internet. Two difficulties have to be resolved: firstly, keeping the same functionalities, even if the server shut down, or if new encryption protocols appear, etc., and secondly, adding new functionalities. Also, for each kind of maintenance, the maintenance system has to know *when* to start such maintenance procedures, *what* to do and *how* to do it. Last, validation of such process would be useful.

Since systems are more and more complex, it is definitely easier to modify part of them instead of challenging their global architectures, which are not agent-based. Therefore, we have to introduce agents into such subsystems, without modifying their architectures, at least at the beginning. In order to do so, we have to develop new kinds of algorithms where MAS is not the main concept used.

To try to solve all of such difficulties, the CRIM laboratory has developed a family of agents, called *Guest*, which are able to run and migrate without modification between different kinds of servers, including CORBA[1]. Also, using plug-ins, a *Guest* agent can, while running, gain or remove pieces of code. For example, such plug-ins could be an implementation of a specific communication protocol, like extraction of data from Web pages written in HTML. We are also working on mechanisms to allow agents to maintain themselves by using meta-modeling and validation. Lastly, we have started to implement protocols as used on peer-to-peer applications to enhance such systems by using isolated agents.

2 Agents able to run anywhere

Until now, an agent could only run on its own platform. For example, an Aglets agent can only use an Aglets based server, not a Grasshopper based one. And vice-versa is also true. In the case of limited multi-agent applications, this is not a real problem. However, future agents running on Internet based applications will have to be able to adapt themselves to heterogeneous servers provided by different partners.

This problem of interoperability could be solved by the use of specific standards [1] [2]. However, to deal with systems that are not agent friendly, other approaches have to be used, such as the use of converters between platforms [3] and the creation of a universal interface (middleware approach).

[1] Corba provided by Corbix from IONA (in a restricted way) and by Java version 1.4.

We choose the last one[2]: we implement our universal agents [4] by using interfaces, more precisely by providing an intermediate layer called *Guest* between our *Guest* agents and the targeted servers (all Java based). This layer is a two-sided entity: on one side, the *Guest* API that is visible to an application programmer, and on the other a server-dependent layer, which we provide. In that way, our agent will be able to run and eventually talk to native agents or objects on these different servers while maintaining the same functionality thanks to interfaces (one per server). This does not imply that all of the servers need to integrate these *Guest* interfaces: it is only when a *Guest* agent reaches a server that the *Guest* Java classes need to be downloaded from a specific location by the Java Virtual Machine (JVM) without modifying the server behavior. This method does not impose a new standard; more precisely parties who want to use the generality of the *Guest* API do not require that *Guest* become a standard. This point is essential to the success of our approach.

A *Guest* agent has therefore two facets: the first one is specific to the platform expected to carry the agent, the other one is independent of any platform. As a result, when a *Guest* agent moves from one platform to another, it has only to change dynamically its platform-specific facet while maintaining its internal status. More precisely, our solution consists in designing a *Guest* agent modeled as the sum of two interconnected agents (when the server is an agent platform), or an object and an agent (for instance when we use CORBA as a server). The first one (so-called native agent / object) inherits from the agent / object class of the targeted agent platform[3], which receive CPU access from the multi-agent platform or by an *ad hoc* thread. The second one (so-called universal agent with the purpose of implementing the agent function) inherits from the universal agent class *Guest*. It is the only part of the agent that will move between servers when the agent migrates[4]. Consequently, our agents are simultaneously perceived by servers as being native and by their originators as being platform-independent *Guest* agents. Our model requires only slightly more resources than a native model: memory consumption and CPU overhead are limited, and are far from being doubled.

A critical constraint has to be mentioned: all the selected servers must run on top of Java Virtual Machines. When a composite *Guest* agent migrates, the generic part will be handed over to the targeted JVM right after the native part is created on it. Upon completion, the composite agent will be removed from its originating platform, thus

---

[2] The same approach is used by CoABS [5] but only to move agents from one agent platform to another. So, they use two kinds of layers: one on top of platforms to a common API, and a from the API to a specific kind of agent. As we know, the CoABS approach only solves the problem of interoperability between platforms; it does not allow "universal agents" to run anywhere.

[3] Since some multi-agent platforms do not provide their source code, rewriting the code of the basis agents is usually impossible. Therefore a *Guest* agent could be send to a new server without modifying this one, but adding new bytecode.

[4] As is the case on most of agent platforms, the current state of the running agent will not been saved and restored when the agent migrates from one server to another. It is the duty of the agent programmer to solve this problem by using specific methods, which are called just before and after the migration.

ending the migration process. One of the conditions for the agent migration is to ensure that the *Guest* agent is serializable.

As mentioned above, our *Guest* agents must have *Guest* interfaces adapted to specific servers in order to be operational. *Guest* interfaces already exists on these Java based agent platforms:

1. ASDK, Aglets Software Development Kit [6], originally developed by the IBM Tokyo Research Laboratory.
2. (Concordia [7] by Mitsubishi, although this was never totally integrated to *Guest* due to a lack of new versions compatible with Java 1.2.)
3. CorbaHost, our own Java server implementation on top of CORBA (Orbix by IONA [8]) and on top of CORBA provided by Java 1.4.
4. Grasshopper [9], a commercial product of the German firm IKV++, which is the first multi-agent platform supporting MASIF [2].
5. Jade [10], an open source platform, which is Fipa compliant[5].
6. Voyager [11], a commercial product of ObjectSpace.

In the case of the CORBA world, which is not agent friendly, we adopted a slightly different implementation. We developed two kinds of CORBA objects: one plays the role of an agent server and provides the service of agent management, while the other encapsulates an agent. These two kinds of CORBA objects are combined into a CorbaHost system.

Such diversity of systems already accommodating *Guest* agents seems to prove that this approach could be extended to most existing and future systems[6].

3 Polyglot Agents

Running agents on a variety of systems is not sufficient. They also have to be able to interact with such systems. The natural way to achieve this is by using the usual access provided to them, through standard API's described by protocols (like HTTP) and languages (like SQL). There is a large number of different API's. So trying to build agents that are able to know all of them is impossible. However, what we have to achieve is to provide to such agents an internal representation of information that is system independent, an internal API that is as universal as possible, a set of interfaces to the main existing systems, and lastly a way to update or integrate dynamically new API's.

Until now, *Guest* agents have had access to information provided by Web pages (see following sections), to Gnutella protocol and services (peer-to-peer approach),

[5] We specially choose this platform to provide Fipa communication capabilities to our *Guest* agents.

[6] As far as they are based on Java, in the specific case of *Guest* agents. However, the same approach could be use for other object oriented language that accept dynamic loading of code and objects.

CORBA Objects, etc. Based on such a variety of interactions, we are currently working on a unified way of dealing with all of them.

3.1 Ontologies to introduce Semantics

Information, such as on a Web page, is not always presented in the same way. Due to this fact, data extraction and exchange is not an easy task if different actors (producers or consumers of information) have not agreed on the semantics of data. This is particularly true in the case of autonomous agents seen as bridges between heterogeneous systems. Consequently, agents need to use a common model for data regardless of their origins. Such a model provides a way to ensure a good understanding and integration of exchanged data.
Ontology appears more and more the way to provide such common model [12]. Ontology is a way to decompose a world into objects, and a way to describe these objects. It is a partial description of the world, depending on the objectives of the designer and the requirements of the application or system. For each domain, there may be a number of ontologies[13]. The use of ontology differs from one application to the next, as do its design and its formalism of representation.

3.2 How to get information from web pages?

The Internet contains more and more Web pages with dynamic and frequently updated data. While a search engine provides useful help for users to identify relevant information, it cannot be used to "understand" the semantics of the results and to obtain reliable data. This is mainly due to the lack of precision and standard formalism in presenting data and because HTML is a visual formatting language. On an other hand, dynamic data (such as weather forecasts, stock exchange information, etc) are more and more required by automated processes such as software agents, and the need is growing to find ways to extract data so they can be fully exploited by agents. This is why we have developed our own extraction engine, which can be integrated into the *Guest* agents.

As we have already said, data on the Web are usually included in HTML pages, and they do not correspond to a given schema. While a human user can understand the data in a Web page display, it is impossible for a machine to do so. Therefore, extracting data from Web pages requires some knowledge of both their structure and contents. There are mainly four approaches to deal with:

1. The first approach relies on natural language processing (NLP). It is known that current NLP is not accurate and powerful enough to recognize the contents of unrestricted web pages and to extract trusted and reliable data. Therefore, this approach has only been used in some limited areas. Manual validation being usually required (as for search engines), use by autonomous an agent is today impossible;
2. The second approach is based on web pages that use some semantic markers (or tags). The limitations of such approach are well known: since the markers are personalized, they can hardly be generalized [12]. In fact, few such pages are available;

3. Currently, an initiative of Semantic Web [14] is geared towards the creation of a web structure that more readily recognizes the semantics of Web pages. The method currently under investigation consists in defining a general ontology of meta data on semantic contents. However, few actual Web pages use such markers;
4. There is a fourth manner to solve the problem witch is based on wrappers or extraction rules to extract data [15][16][17][18][19]. A wrapper is a piece of code generated manually or by means of a tool, which allows retrieving data from a source (Web page).

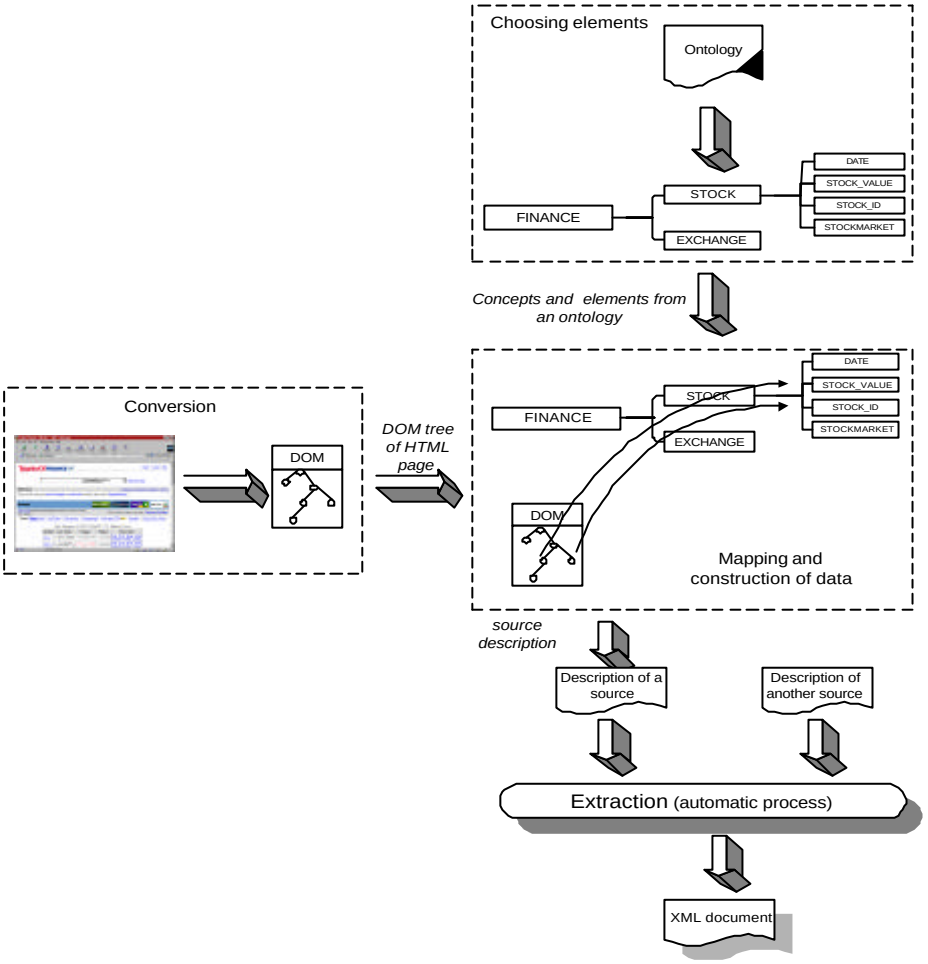

Fig. 1: Global extraction processus

Our approach, implemented into a user-friendly graphical tool, called WeDaX [20], is close (but also different) from the later approach. We will make use of ontology to model the data to be extracted and a source description (semi-automatically gener-

ated), which indicates how the desired data can be extracted (**Fig. 1**). In particular, we will focus on data extraction from web pages that present constantly changing data, but with a fixed structure (*e.g.* stock exchange quotes). The data in a web page is first converted into XML[7], and then mapped with the data model. The definition of the data model and the mapping are done manually. Then an automatic process can be carried out repeatedly to perform the real extraction tasks. The final results are XML documents that contain standardized and queriable data sets.

Since the data extracted by WeDaX are accurate, autonomous agents like the *Guest* agents can use them. However, because the algorithm we use needs specific kind of HTML structures like tables, the kind of data we can extract is limited.

4 Adaptive agents

In an open, evolving multi-agent world, algorithms and services will be put into use during the life cycle of an agent or system. For example, a new compression algorithm may be used to compress messages, or a new kind of cryptographic signature may be released, during the life cycle of agents. Moreover, it's possible that at one moment in the life of an agent, changes in the environment require that the agent modify its execution model. One example is the deployment of new servers to handle an existing application with new load balancing capabilities. In this case agents need to change to a distributed model. To adapt to these evolution and openness of the environment, our agents need to be able to apply "on the fly" these new algorithms, services and models, without having to restart any part of the system. More precisely, we want to be able to develop "new capabilities" in an agent-independent way, and then allow the *Guest* agents to use them *on demand*.

4.1 The plug-in framework

Guest agents are currently able to operate on heterogeneous servers, thanks to the interface layer on top of these ones. However, this interface must be fixed at runtime and is not suitable for dynamically adding new capabilities to an existing agent: any modification to this interface requires the restart of all the agent servers on which the *Guest* agents are running. Therefore, we need a more convenient way to solve this problem. Our solution is to embed in the kernel of the *Guest* agent a framework called *plug-in*. A plug-in is a component (compiled code of one or some objects) that can be

[7] Documents in HTML do not allow for direct querying. In addition, there may be errors in HTML structures. Therefore, we first convert the HTML document into XML by following their hierarchical structure [21]. Possible errors are corrected using HTML TIDY [22] and W4F [23] (correction and transformation module of W4F). Once a web page is transformed into XML, portions of data can be easily accessed using a DOM (Document Object Model) parser [24].

dynamically loaded by an agent and which immediately offers some new services. An agent then consists of two parts: a kernel, which is fixed and a dynamic set of plug-ins. A plug-in can perform any of the following three types of actions:

1. Observe changes in the state of the associated agent and possibly prevent these changes in some cases (migration or deactivation of the agent);
2. Observe the communications of the associated agent (sending and receiving a message) and possibly intercept and modify an incoming/outgoing message;
3. Offer a library of new services to the agent.

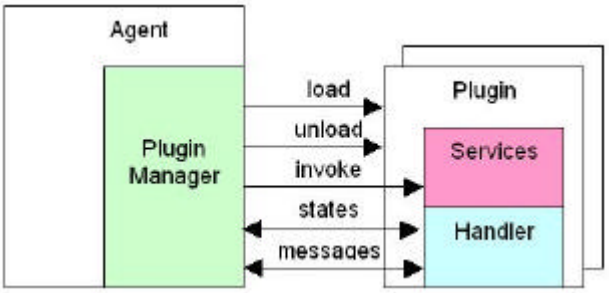

Fig. 2: The plug-in framework

Therefore, a plug-in can not only add new functionalities to an agent, but also change behaviors of the agent. When a plug-in needs to manipulate the incoming/outgoing message or the states of the agent to which it is attached, it must provide the corresponding interceptor and then subscribe to the Plug-in Management Service. If there are several interceptors of the same type, one solution is to chain and then activate them one after another. In fact, this mechanism is not a mere chaining, but is more sophisticated. Upon loading, a plug-in exports to the agent its *activation function*. This function is a filter, applied to incoming messages and events, deciding on which ones the plug-in offers its services. For example, a dedicated decompression plug-in would ask for activation only when compressed messages are received. It would then ask for activation for each potentially encapsulated compressed message. Any given plug-in can be used any number of times during the processing of a message or an event. This allows us to provide new capabilities (for instance compression and cryptography), even if we don't know in advance the proper layering of the message by the other agents (do they first sign, and then compress? Or do they compress, and then sign?). This is a very important capability in the kind of open environment we developed *Guest* for.

In order to reinforce the security of the framework and to simplify the development and the use of our plug-ins, each plug-in is divided into two sub-components:

? A "private part", called the *handler*, which is responsible for the first two types of actions: observe/intercept the state and the communications of the agent;
? A "public part", which provides new services.

In the one hand, this framework enables a *Guest* agent to modify dynamically its capabilities and therefore adapt to the evolving environment by allowing a plug-in to be associated with an agent at any time during the agent's life and removed whenever the agent no longer needs it. For example, one agent can “learn" to compress/decompress data by loading a compression plug-in. When it no longer needs these functions, it can simply “forget” it by unloading this plug-in. The agent can even “upgrade” its compression technique by changing the current compression plug-in for a more sophisticated one. On the other hand, reuse and the sharing of the development effort can be archived using this framework. A plug-in can be developed by a third party, independently of the development of agents. The deployment of these plug-ins is also flexible: one plug-in doesn't have to be bundled in the same package with the agent but can be downloaded from an Internet site. In many ways, a plug-in can be used as an *off the shelf component*.

Plug-ins have already been used for a larger panel of features provided by *Guest* agent, as for example:

? *Visualization of the Agent Management Service*: we have already developed a utility with a graphical interface to visually observe *Guest* agents on different native platforms (**Fig 3**). This was done by creating a plug-in that intercepts the changes of agent's state during migrations and informs the GUI to visually reflect these changes on the screen. Consequently, an agent can be observed simply by associating with it this plug-in even if this agent was not initially supposed to offer this service;
? *Groups of interest*. By using an *ad hoc* plug-in, *Guest* agents are able to broadcast messages regarding a specific topic only to agents that have previously shown their interest into that topic;
? *Validation of interactions.* Plug-ins can be added to an agent to intercept all of its messages to produce a log-file of all of them. Such log files are currently used to perform verifications of interactions between agents;
? *Virtual hierarchies*. By re-routing messages of a “child” agent to its “father”, it is possible to produce virtual hierarchies of agents.

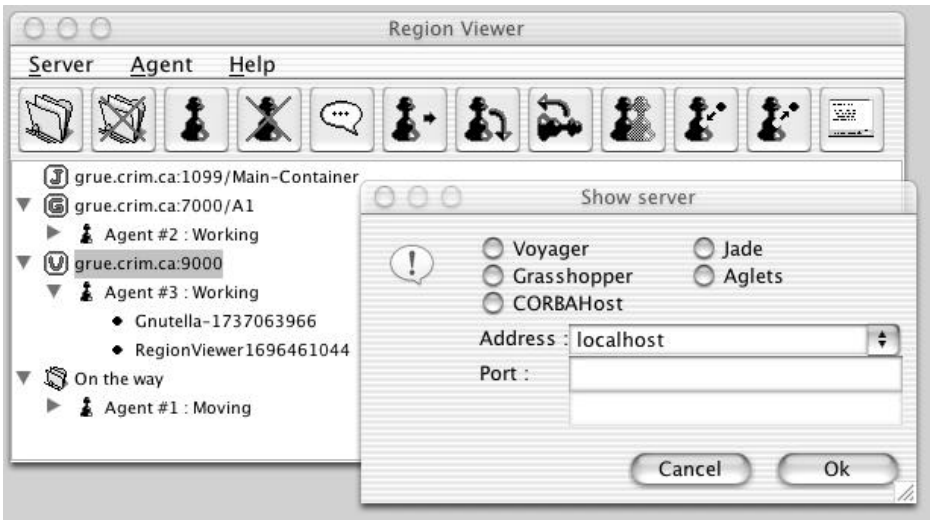

Fig 3: *Guest* visual interface

4.2 Adaptation by meta-modeling

Sometimes, changes in the environment require agents to change their execution model to adapt to new constraints. Therefore, agents need to be able to change "on the fly" from one model to another. Our approach to solve this problem is to develop a model of the control in different agent models (**Fig. 4**), which can be called meta-model of agent control, and which aims at facilitating dynamic transformation from one model to another one. More precisely, our solution consists of the following steps:

? Define a meta-model of agent control allowing a uniform representation of the control of different agent models and dynamic transformation between these mo dels;
? Validate these transformations.

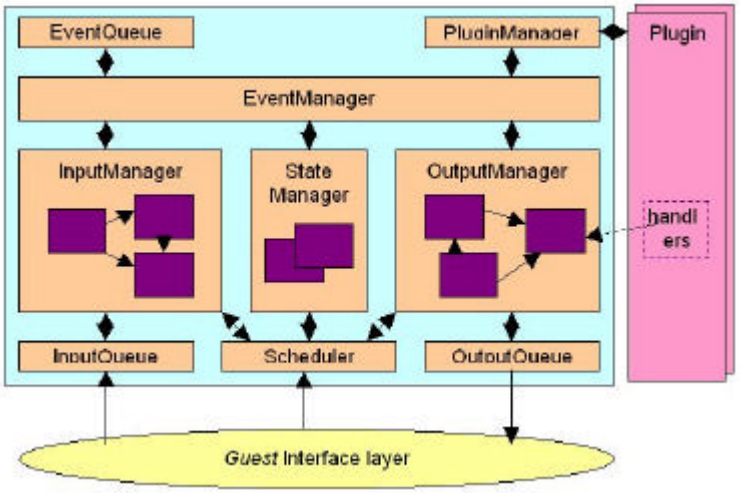

Fig. 4: The meta-model of agent control

The principle of our meta-model is to classify all the modules of one agent model into different groups by their roles and then to provide a meta-module to manage each group. From the agent's traditional execution cycle: *perception – deliberation – action*, all the modules of one agent model can by classified into these three main categories. The coordination between these groups can be realized using either a control-driven or a shared data space approach [25]. We have chosen the later, because of the centralization of the modules and the complexity of the coordination forms. Each module is associated with a *configurator*, which is responsible for managing the configuration of this module (connections to other modules, kinds of events this module is interested in, etc.)

The dynamic transformation from one model to another consists of the replacement the modules of the first one by those of the second one and the reconfiguration of the link between the new modules, using the configurators presented above.

4.3 Validation of interactions between agents

We developed a formalism for the specification and validation of a multi-agent system. This formalism uses the CATN model [26] for the specification of a multi-agent system, and the GOAL model [27] for the validation phase. CATN focus on communication protocols between agents, and allow us to define easily interaction patterns among agents. These patterns are the inputs of a CATN manager module, which defines the behavior of the agents running in the multi-agent system. For the validation phase, we define properties to be checked as observers in the GOAL language[8]. This is a model-

[8] A GOAL observer implements a finite automaton with accepting states.

checking approach, working on a lattice built upon the collected trace of all the agents in the system. We build by hand the list of properties to be checked against the lattice, and our current work is the development of a mapping between the CATN formalism for interaction-pattern definition into the GOAL formalism for property-checking.

5 Conclusion & perspectives

Providing agents that are able to deal with various systems is the main goal of the *Guest* project. So far, we succeeded to the extent that agents can run and move between heterogeneous systems, even if such systems are not able *a priori* to welcome agents. Also, the plug-in framework allows *Guest* agents to acquire new features or update existing ones. Using such plug-ins, some non agent-based protocols and languages are already available to *Guest* agents, such as extraction of information from Web pages, Gnutella peer-to-peer services, etc. Meanwhile, to provide agents that will be able to evolve in real open systems, more work has to be done. Currently, we are working on a unified method of communication between heterogeneous systems, on mechanisms to allow agents to adapt themselves to modifications of their environment, on the description of agent interactions based on CATN, tools to verify the interactions between agents, etc. Lastly, we are also using *Guest* agents in a project called Geref [28], which consists in the management of distributed Workflows.

6 Acknowledgements

This work was done at the CRIM laboratory and mainly funded by it. We have also to acknowledge the efforts of Nekrouf Ziani for HostCorba implementation and of Ivan Guentchev for his experiments on Jade, who were (along with other students who worked on this project) supported by Laurent Magnin's personal Research Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

The work regarding WeDaX was initially supported by a scholarship of AUPELF to Hicham Snoussi, and a complementary scholarship from NSERC.

References

1. Fipa, Foundation for Intelligent Physical Agents, http://www.fipa.org/
2. MASIF: Mobile Agent System Interoperability Facility, Object Management Group OMG, http://www.fokus.gmd.de/research/cc/ima/masif/index.html
3. Tjung D., Tsukamoto M. and Nishio S., A converter Approach for Mobile Agent System Integration: A Case of Aglet to Voyager, Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999, pp. 179-195

4. Magnin L. and Alikacem E.H., *Guest*: Multiplatform Generic Agents, Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999, pp. 507-514
5. Control of Agent Based Systems, http://coabs.globalinfotek.com/
6. Aglets Workbench, http://aglets.sourceforge.net/
7. Concordia - Java Mobile Agent Technology, http://www.concordiaagents.com/
8. Orbix by IONA, http://www.iona.com/
9. Grasshopper, Grasshopper by IKV++, http://www.grasshopper.de/
10. The Java Agent DEvelopment Framework, http://jade.cselt.it/
11. Voyager, Voyager by ObjectSpace, http://www.objectspace.com/voyager/
12. Atzeni, P., Mecca, G. and Merialdo, P., To Weave the Web - In Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97), 1997
13. Bezivin, J., Les nouvelles convergences : Objets, composants, modèles et ontologies, JICAA'97, Roscoff France, Mai 1997
14. W3C-Semantic Web, http://www.w3.org/2001/sw/
15. Knoblock, C. A., Minton, S., Ambite, J. L., Ashish, N., Modi, P. J., Muslea, I., Philpot, A. G. and Tejada, S., Modeling Web Sources for Information Integration. Proceedings of the National Conference on Artificial Intelligence, Madison, 1998
16. Ouahid, H and Karmouch, A., An XML-Based WEB Mining Agent, Proceeding of MATA'99, Ahmed KARMOUCH and Roger IMPEY eds., World Scientific, Ottawa, 1999
17. Ling Liu, Calton Pu, Wei Han. "XWRAP: An XML-enabled Wrapper Construction Sy stem for Web Information Sources", Proceedings of the 16th International Conference on Data Engineering (ICDE'2000), San Diego CA (February 28 - March 3, 2000)
18. David Buttler, Ling Liu, Calton Pu. "A Fully Automated Extraction System for the World Wide Web." IEEE ICDCS-21, Phoenix, Arizona (April 16-19, 2001)
19. Sahuguet, A. and Azavant, F., Building light-weight wrappers for legacy Web data-sources using W4F, International Conference on Very Large Databases (VLDB), Edinburgh - Scotland – UK, September 7 - 10 1999
20. Snoussi, H., Magnin, L. and Nie, J.-Y, Toward an Ontology-based Web Data Extraction, The AI-2002 Workshop on Business Agents and the Semantic Web (BASeWEB) held at the AI 2002 Conference (AI-2002), Calgary, Alberta, Canada, May 26, 2002
21. W3C, http://www.w3.org
22. Raggett, D., HTML Tidy, http://www.w3.org/People/Raggett/tidy/
23. Sahuguet, A. and Azavant, F., Looking at the Web through XML glasses, Conference on Cooperative Information Systems CoopIS'99, Edinburgh Scotland, September 2-4 1999
24. Document Object Model,http://www.w3.org/DOM/
25. Papadopoulos G. A., Coordination of Internet Agents: Models, Technologies and Applications, Chapter : Models and Tecnologies for the Coordination of Internet Agents: A survey, Springer-Verlag, 2001.
26. Zavala R. Laura, Carreño Araceli A., Lemaître Christian, CATNAgentToolkit: Una plataforma para el diseño y ejecución de sistemas multiagentes, Laboratorio Nacional de Informática Avnzada, México
27. H. Hallal, A. Petrenko, A. Ulrich, and S. Boroday, "Using SDL Tools to Test Properties of Distributed Systems", the proceedings of FATES'01, Formal Approaches to Testing of Software, A Satellite Workshop of CONCUR'01, Aalborg, Denmark, August 25, 2001

28. Arnaud Thiefaine, Thang Viet Pham, Laurent Magnin, Gil Blain, De la description à l'exécution de systèmes multi-agents: Approche Metagen appliquée aux agents *Guest,* JFIADSMA 2002, 28-30 october 2002, Lille, France