Programming and Executing Telecommunication Service Logic with Moorea Reactive Mobile Agents

Bruno Dillenseger[1], Anne-Marie Tagant[1], Laurent Hazard[2]

France Télécom R&D, DTL/ASR
[1] 28 chemin du Vieux Chêne, BP 98, 38243 Meylan cedex
[2] 38/40 rue Général Leclerc, 92794 Issy Moulineaux cedex 9
{bruno.dillenseger, annemarie.tagant, laurent.hazard}@rd.francetelecom.com

Abstract. In the context of advanced telecommunication service execution environments, this paper presents a reactive mobile agent platform, based on a synchronous programming model, a flexible object request broker, and OMG's MAF specifications on mobile agent systems interoperability. The paper details the specific design of the platform, and shows how it addresses key issues such as scalability, reliability, transparent mobility and interoperability. Finally, the paper shows the use of Moorea agents for executing telecommunication service logic in the ITEA Athos European project's Enhanced Call Server architecture, featuring transparent and dynamic distributed system reconfiguration.

1 Introduction

As of today's research topics, involving (mobile and/or intelligent) agent technology in telecommunication infrastructures is a well-known trend (e.g. see a digest of several European projects in [12]). It has been used already in several experiments ranging from information infrastructure to network management and active networks. Interest in mobile agent technology mainly arises from its ability to handle in a unified way a variety of issues [10], such as code deployment, autonomous adaptive routing, network bandwidth saving, disconnected operations...

The topic of this paper is the use of mobile agent technology in the context of a distributed execution environment for telecommunication applications. While many practical applications of mobile agent technology exploit only a subset of the paradigm (e.g. single-hop, weak mobility), our approach makes a complete utilization of the advanced properties of the Moorea platform. The development of Moorea springs from the consideration that mobile agent technology often comes with several drawbacks, such as bad scalability, cost and complexity of agent activity transportation, unreliable agent communication and lack of interoperability. Moorea is a Java mobile agent platform implementing the Mobile Agent Facilities specification [13]. It comes with a peculiar reactive model for agent communication and activity, enhanced with distribution support and transparent mobility features.

This paper first describes the Moorea platform, details its specific properties and compares them to existing platforms. Then, the paper describes the use of Moorea in the field of telecommunication service execution environments, through the example of an Enhanced Call Server developed in the context of the Athos European project.

2 Moorea Overview

2.1 Moorea architecture

Moorea ("MObile Objects, REactive Agents") is a 100% Java reactive mobile agent platform. As shown by Fig. 1, its architecture combines:

- a reactive object model and execution kernel (*Rhum and Junior* [11]),
- with a Java mobile object framework (*SMI* [7]) implementing OMG's Mobile Agent Facilities specification [13] (MAF, also known as MASIF),
- on top of a flexible Object Request Broker (*Jonathan* [21]) offering both a Java-RMI and a CORBA personality (API),
- extended with transparency to mobility support.

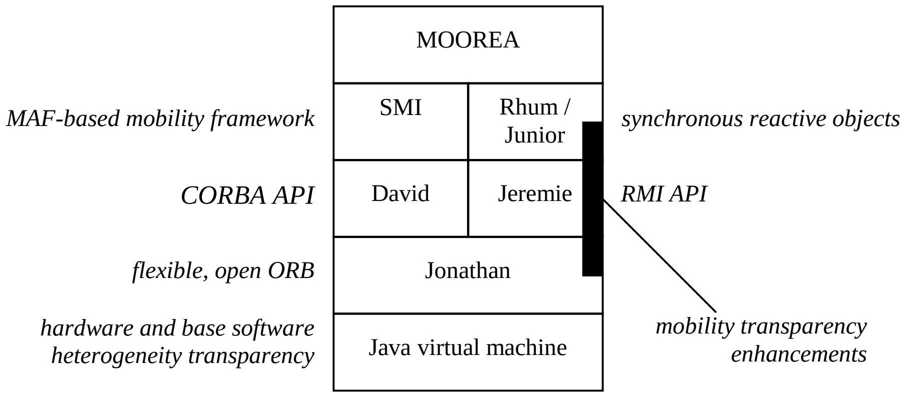
MOOREA
SMI
Rhum / Junior
David
Jeremie
Jonathan
Java virtual machine
MAF-based mobility framework
CORBA API
flexible, open ORB
hardware and base software heterogeneity transparency
synchronous reactive objects
RMI API
mobility transparency enhancements

Fig. 1. Moorea's software architecture is based on the integration of a MAF-based mobile agent framework (SMI) with a reactive distributed object environment (Rhum).

2.2 Key concepts

2.2.1 Reactive agents and events

Moorea's agent model is based on Rhum's distributed reactive object model (Fig. 2). The basic processing procedures and internal data of an agent is embedded in a reactive object which controls its *behavior*. A behavior describes the agent's activity in a synchronous language - Rhum - featuring high-level constructs to define parallel branches, conditions, synchronization, loops... Agents are *reactive* in that the execution of the behavior may both depend on (or *react to*) and generate *events*.

Agents are created, hosted and executed by *agencies* (agent systems in MAF terminology). Events are generated by agents or by their host agency. Events may hold values, and may be either locally visible by all the agents in a given agency, or specifically sent to a given agent, whatever local or remote. The exact semantics of events is given in section 2.2.2. Agents are designated through their *reference*, which may be used for managing them or for sending them events.

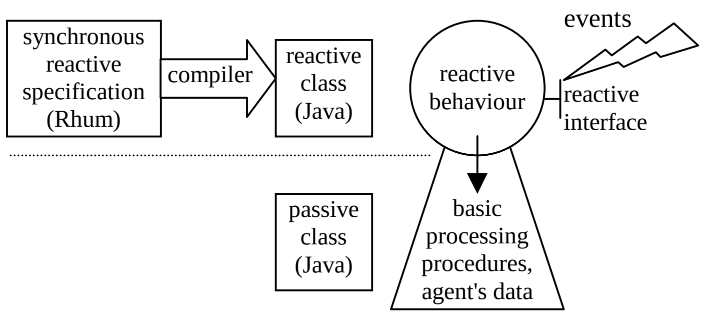

Fig. 2. The behavior is compiled from a Rhum program to a Java reactive class, embedding the associated passive object. Events are mapped to methods of the object's reactive interface.

2.2.2 Focus on Synchronous Programming model

Derived from Esterel [3], synchronous language Rhum slightly modifies its semantics to avoid causality problems and to allow dynamic program composition. However, the synchronous execution principle remains: in such an execution model, time is sliced into logical instants, which define the lifetime of an event and the reaction semantics (Fig. 3): (1) an event is present during an instant if and only if it is generated during this instant; (2) reactions to an event are run in the same instant; (3) an event may trigger reactions only once per instant, whatever the number of times the event has been generated during this instant. An instant ends once all reactions are terminated or stopped. A reaction stops by waiting an event which is not present in the instant, or by explicitly waiting next instant.

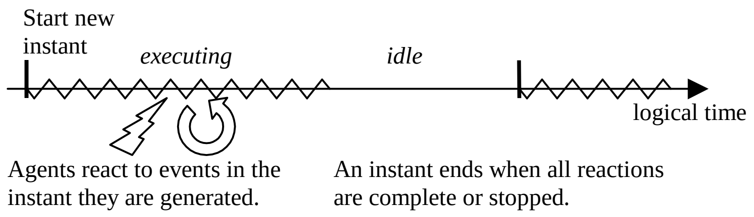
Start new instant
executing
idle
logical time
Agents react to events in the instant they are generated.
An instant ends when all reactions are complete or stopped.

Fig. 3. A synchronous execution is split in instants where agents react to and generate events.

Each agency contains an engine to locally manage instants, events and reactions. Agencies are independent *reactive domains* (for efficiency reasons, sequences of instants of distinct agencies are fully independent).

2.2.3 Scalability aspects

The execution model is based on reacting in sequence (i.e. with no actual parallelism) to events. This model both saves processing time (no preemptive scheduling and no context switching) and avoids complex code related to concurrency and consistency management. This model thus enforces good scalability, in terms of number of agents per agency. A scalability comparison of Rhum with thread-based agents is given in [8]. The results show that the number of reactive agents does not impact the system normal functioning, while allowing a greater number of agents. At the present time, we can run a million reactive agents on a single PC.

2.3 Strong Mobility Support

In the mobile computation community, Fugetta *et al* [9] define the concept of weak and strong mobility, to express the fact that a computation may be more or less disturbed by mobility. As a matter of fact, a running process uses local computing resources, which, after a move, either would have to be accessed remotely, or would have to be replaced by local similar resources in the "same" state.

Obviously, today's operating systems do not support such features. As a consequence, support for transparency to mobility must be provided by upper software layers. Moorea claims a strong mobility support for two reasons: (1) execution is not disturbed and (2) communications are not disturbed.

2.3.1 Transparency to Mobility for Execution

Moorea agents' behavior is represented by a reactive program, whose execution is split into instants. The beginning of an instant is triggered by the reactive domain, and the end of an instant is reached when no agent reacts any more. At the end of an instant, the state of agents is stable, well defined and easy (not costly) to transport. Moorea takes advantage of this property, by actually performing moves once the end of instant is reached. A moving agent is frozen and transported with its behavior and full state, and then resumed in the new reactive domain (agency), in a new instant.

Note that the combination of mobility and reactive model considerably limits mobility overhead. This approach must be compared to thread-based mobile agents, whose mobility is either disturbing (the execution state is reset to some default state after move), or complex and costly (mobility includes execution stacks). On the contrary, Moorea's agent model allows a low-cost serialization and transport of agents' execution state (an agent behavior is equivalent to a state machine). Moreover, the synchronous reactive model offers clear mobility semantics, which is not always the case of thread-based mobile agents introducing (sometimes unclear) side effects to the programming semantics. Besides, the programmer is freed from concurrency management burden, as well as inconsistency and deadlock threats (see 2.2.3).

2.3.2 Transparency to Mobility for Communication

Since Moorea reactive model tightly couples activity with communication, transparent mobility must also consider events. While environment events remain purely local, targeted events should always follow the target agent, without being lost, even during the agent transportation timeframe. This transparency is achieved by a combination of two well-known techniques, namely forwarding and naming service:

– forwarding consists in replacing the moved agent by a forwarder object that forwards invocations to the new location. Basic forwarding can lead to a long *reference chain* (multiple hops), relying on every visited agency to keep running, which is not always assumable for mobile agent applications;
– a naming service – or *relocator* – associates a name to a location-dependent distributed object reference. By updating this information after each move, mobile agents can still be located. The major drawback of this technique is that it introduces a central authority and bottleneck.

Moorea combines both techniques while seriously limiting these drawbacks. First of all, the reference chain maximum length is limited to one indirection, by having forwarders directly get the new agent location from the relocator. Moreover, the forwarder updates the reference in the clients, which reduces the relocator "bottleneck effect" by preventing clients from invoking the relocator. Lastly, there may be several relocator servers, defining their own name spaces, and avoiding to enforce globally unique agent names.

Let's assume that an agent is moving from agency A to agency B, and that other agents send it targeted events:

- during the move, and as long as the agent has not been reinstalled in agency B, events are buffered in agency A;
- agency A gets the new agent reference from the relocator, forwards events to agency B, and gives the new agent reference to the senders of the events;
- if agency A is unreachable (e.g. it has been shut down), the client directly gets the new reference from the relocator.

These transparency features are implemented by the stubs in the underlying middleware (i.e. Jonathan and its RMI personality Jeremie, and Rhum) [14]. They are not fully specific to Moorea, and have been reused in another work in progress, to support object mobility in Jonathan's CORBA personality (David).

3 Related Work

3.1 Middleware Approach, Interoperability

Moorea follows a middleware approach, which was not the case for historical mobile agent platforms. Not only Moorea is built on open ORB Jonathan, but its mobility feature is fully CORBA-based and MAF compliant thanks to the underlying SMI framework [7]. Moorea agents and agencies communicate through the ORB, using common services such as naming service. Previous similar approaches, and their benefits in terms of code reuse and agent interoperability, have already been described in [6]. Below SMI, other known mobile agent systems implementing MAF are Grasshopper [20], SOMA [23]; version 1.1 of Aglets [17] plan to be MAF compliant.

Besides MAF specification, the activities of the Foundation for Intelligent Physical Agents consortium (FIPA [19]) and OMG's Agent Special Interest Group [22] (including a mobility working group) show that interoperability is considered as one of the key issues for the success of agent technology.

3.2 Transparency to Mobility

Most of mobile agent platforms take care of transporting the agent's code and "static" state when moving an agent, but very few of them are able to transport the execution state, i.e. to resume the agent execution at the very point of its execution. Dartmouth College's AgentTcl/D'agent [10] and General Magic's Telescript [16] support such strong mobility. The problem with strong mobility is that it is typically complex and costly, especially when agent activity is based on threads (e.g. see work on Java thread migration in [15]).

An efficient alternative of the "move anytime" approach is the "move sometimes" approach, which means that the agent activity can be interrupted, frozen, transported and resumed only at certain well identified points. In the case of Moorea, these points identified by the ends of instant (i.e. when the agent is waiting for an absent event or for next instant, see 2.2.2). After a move, an agent transparently resumes its activity in a new instant, at a new location. This semantics is clear - and "natural" in some way -, whatever the behavior complexity and the number of active loops or parallel branches.

As a consequence of this approach, the transparent mobility support is provided at low cost, by avoiding to freeze and transport "expensive" items like running threads.

3.3 Formal "Behavior"

Thanks to the separate programming of the agent global behavior and basic computing procedures, Moorea exhibits the agent activity logic, with its parallel branches, synchronization points, event waiting. This high level activity representation is not only the key to low-cost, portable mobility, but also a key to easy and reliable agent programming, since it prevents the computing procedures from handling monitors, semaphores, locks, etc., for managing synchronization and concurrency issues. Moreover, it opens the way to simulating and testing, and even probably to proving execution properties, as it is already the case with Esterel programs [4].

The Bond agent system [5] follows a very similar approach: agent activity is controlled by a multi-plane state machine, generated from a description in a dedicated language called "BluePrint". The basic processing procedures are implemented by a set of so-called "strategies" objects, equivalent for Moorea's passive object. Unlike Moorea, it is not based on a synchronous programming model.

Being also based on synchronous reactive objects and a dialect close to Rhum, Rejo is the most similar approach [1]. This on-going work goes further in the integration between Java and the reactive language, since both the reactive behavior and the Java code are mixed in the same file. Usage will tell what is the most convenient between separating basic processing and activity skeleton with Moorea, or having a unique, unified agent definition with Rejo. As a major difference, events in Rejo are basically local, while Moorea object references support distribution.

4 Application to a Telecommunication Service Execution Environment

4.1 Requirements of Telecommunication Services

In the ATHOS project [18], Moorea is integrated to a telecommunication service creation and execution environment. This European project is aiming at defining a relevant architecture for such an environment, in order to develop and run services on a bunch of computers linked to telecommunication networks through legacy protocol stacks, typically in an intelligent network architecture. The service creation and execution environment is expected to support several advanced features.

First of all, *distribution* would be valuable, for the execution environment should be highly *scalable* to support a great variety of services (from tens to hundreds) and to handle thousands (or more) of simultaneously active instances of these services. Moreover, the execution environment should be able to be continuously running for a long period of time (at least for months). This requirement implies that it should be possible to *dynamically reconfigure* the execution environment for maintenance issues (e.g. add, reboot or shutdown a node) and to dynamically update software.

4.2 Moorea in the ATHOS Architecture

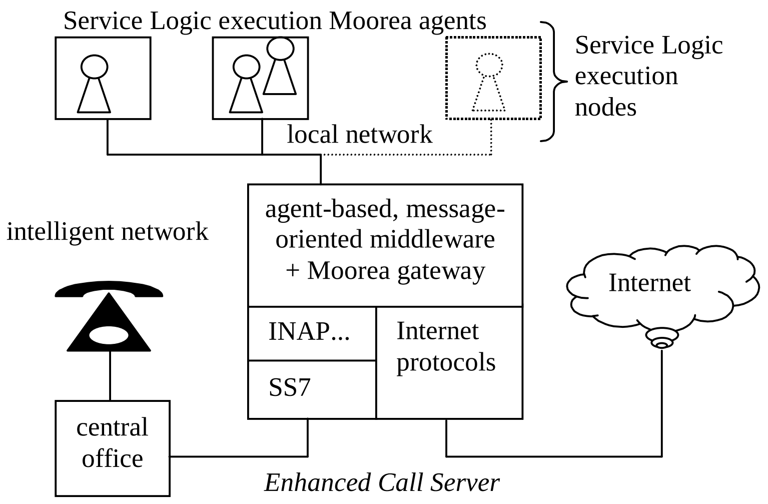
Service Logic execution Moorea agents
Service Logic execution nodes
local network
agent-based, message-oriented middleware + Moorea gateway
intelligent network
Internet
INAP...
SS7
Internet protocols
central office
Enhanced Call Server

Fig. 4. The ATHOS Enhanced Call Server architecture makes it possible to build advanced services, accessing both the Internet and the Intelligent Network worlds, using modern and open information technologies.

The ATHOS architecture is based on the concept of Enhanced Call Server (or ECS, Fig. 4). The ECS embeds the protocols-specific knowledge and software of both the Internet and Intelligent Network worlds. Its static infrastructure is composed of a Unix server running the AAA message-oriented middleware [2] (see also JORAM in [21]) and protocol stacks connected through IP to a gateway giving access to the Intelligent Network. The service logic is executed by Moorea agents on a local network of computers, providing the dynamic infrastructure of the ECS.

4.3 Example: a simple Email Waiting Indicator service

4.3.1 Scenario

A demonstration of Email Waiting Indicator (or EWI) service is developed on the ECS architecture. This service consists in providing a phone subscriber with information about his pending electronic messages. Once the service is subscribed and the information about the email account is provided (typically the IP address of the POP3 server, identifier and password), the user may enable and disable the service by dialing special codes on its phone. When the service is enabled and the user unhooks his/her phone, s/he gets the information about pending messages (e.g. special tone or voice message). These IN events (dial codes, off-hook) are handled by the ECS, which performs the appropriate action.

4.3.2 Implementation

To enable the service, the ECS asks a Factory agent to make a new EWI service logic agent ready. The Factory agent either recycles a passivated EWI service logic agent, or creates a new instance if necessary. Note that the service logic agent may be created and reside in any agency, on any execution node. Then, the Factory agent sends an "activation" event to the service logic agent and provides it with the user's profile holding the necessary data (identification, e-mail account...).

Once activated, the service logic agent sends its reference to the ECS which stores it in the user profile. Finally, the service logic agent starts polling the e-mail server in a generic way, by using an e-mail server encapsulation provided by the ECS, in order to keep the service logic independent from the protocols. The polling period is based on a "tick_1s" event generated every second by a timer agent in each agency.

When the user unhooks the phone, the ECS finds the EWI service logic agent's reference in the user's profile and sends a "signaling" event to the agent. The service logic agent gives the information about pending messages to the ECS which transmits the information through the Intelligent Network to the user in an appropriate manner.

```
behavior service_behavior() {
    loop // endless loop (i.e. until agent is terminated)
          await activation ; // wait for occurrence of activation event
          { activation(activation::arg_0) } ; // invoke passive object
          stop ; // wait for next instant
          do
                loop // poll email account
                      repeat { delay } times // wait a number of ticks
                            await tick_1s ; stop // from a timer agent
                      end ;
                      { polling() } ; // ask mail server for information
                      await reply ; // wait information from mail server
                      { getReply(reply::arg_0) }
                end
                || // parallel composition
                loop // give e-mail information to subscriber when necessary
                      await signaling ;
                      { signaling() } ;
                      stop
                end
          until passivation ; // exit on occurrence of passivation event
          { passivated() } // inform the Factory agent
    end }
```

Fig. 5. The Simple Email Waiting Indicator service logic is executed by a Moorea agent, whose behavior is coded in synchronous reactive language Rhum.

When the user disables the service, the ECS sends a "passivation" event to the EWI service logic agent, which in turn informs the Factory agent and goes idle. The user profile is updated to clear the service logic agent reference. The passivated agent will be recycled by the factory for any later EWI service activation for any subscriber.

Fig. 5 shows the reactive behavior of the corresponding EWI service logic agent.

4.3.3 Advanced features enabled by Moorea

Thanks to the strong mobility support, the service logic agents may be transparently moved from one agency to another, from one execution node to another. This makes it possible to add and remove execution nodes dynamically. We introduced in each agency a Manager agent, whose role is to dispatch every local service logic agent to other agencies before shutting down the agency. This move is transparent to the ECS and the Factory agent, which can still use the same references to deal with, and manage, the service logic agents. In the other way round, new agencies are taken into account on the fly by the Factory and Manager agents to host new or redeployed service logic agents. This is also an opportunity to implement preemptive dynamic load balancing, which is the next step of our work.

Besides these features, we found useful (after an adaptation stage) to use a synchronous language to code the service logic, and not to have to care about concurrency issues when programming the associated passive classes. The reactive behavior is also a good basis for simulation and checking.

5 Conclusion

Moorea is a 100% Java mobile agent platform, based on a standard software infrastructure (Java, RMI, CORBA, MAF), and a distributed reactive object model. These peculiarities improve portability, interoperability and standardized management on the one hand, as well as scalability, transparency to mobility, and reliability on the other hand. We have shown an application of Moorea to a telecommunication service execution environment. Based on synchronous reactive language Rhum, Moorea offers the opportunity to conveniently implement telecommunication services by separating the service logic from the basic computations. Moreover, the transparent, low-cost mobility of Moorea agents provides an opportunity for dynamic redistribution and load-balancing of executing services.

Immediate directions for future work on Moorea deal with scalability characterization and preemptive dynamic load balancing. Further directions include model-checking and dynamic versioning.

Acknowledgement

Moorea is developed in the context of the ATHOS project of the ITEA European research program. Thanks go to Italtel team, for their design of the ECS and their interest in agent technology, and to Serge Lacourte and Laurent Chauvirey for their help on the AAA-Moorea Gateway.

References

[1] Acosta Bermejo, R., "Programming in REJO", *Calculateurs parallèles, special issue* "Evolutions dans le domaine des intergiciels", Hermes ed., 2001.

[2] Bellissard L., de Palma N., Freyssinet A., Herrmann M., Lacourte S., "An Agent Platform for Reliable Asynchronous Distributed Programming". *Symposium on Reliable Distributed Systems (SRDS'99)*, Lausanne (Switzerland), 20-22 October 1999.

[3] Berry G., Gonthier G., "The Esterel Synchronous Language: Design, Semantics, Implementation", *Science of Computer Programming*, 19(2), 1992.

[4] Bertin V., Poize M., Pulou J., Sifakis J., "Towards Validated Real-Rime Software", *proc. 12th Euromicro Conference on Real-Time Systems*, Stockholm, june 2000.

[5] Bölöni L., Jun K., Palacz K., Sion R., Marinescu D., "The Bond Agent System and Applications", *proc. ASA/MA 2000*, *Lecture Notes in Computer Science 1882*, Springer, pp. 99-112.

[6] Dillenseger B., "From Interoperability to Cooperation: Building Intelligent Agents on Middleware", *proc. 2nd International Workshop on Intelligent Agents for Telecommunication Applications*, Paris, july 1998. *Lecture Notes in Artificial Intelligence 1437*, Springer, ISBN 3-540-64720-1, pp. 220-232.

[7] Dillenseger B., "MobiliTools: An OMG standards-based toolbox for agent mobility and interoperability", *proc. 6th IFIP Conference on Intelligence in Networks (SmartNet 2000)*, Vienna, september 2000, Kluwer Academic Publishers, pp. 353-366.

[8] Dillenseger B., Tagant A.-M., Hazard L., Tran Viet H., "Les agents mobiles réactifs Mooréa - une approche réactive pour la transparence à la mobilité et le passage à l'échelle" RSTI-TSI 21/2002 *"Agents et codes mobiles" special issue*, Lavoisier-Hermès ed. p. 1-26.

[9] Fugetta A., Picco G.-P., Vigna G., "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, vol. 24, No 5, 1998, pp.342-361.

[10] Gray D., Kotz D., Nog S., Rus D., Cybenko G., "Mobile Agents: the next generation in distributed computing", *proc. 2nd Aizu Int. Symposium on Parallel Algorithms and Architectures Synthesis*, Fukushima (Japan), IEEE Computer Society Press, 1997, p. 8-24.

[11] Hazard L., Susini J.-F., Boussinot F., "The Junior reactive kernel", *Rapport de recherche Inria 3732*, 1999.

[12] InfoWin project, "Agents Technology in Europe, ACTS Activities", 1999, ISBN 3-00-005267-4.

[13] Object Management Group, "Mobile Agent System Interoperability Facilities", *TC document orbos/97-10-05*, 1997. Revised in "Mobile Agent Facilies", *formal/2000-01-02*.

[14] Tran Viet H., "Gestion de la mobilité dans l'ORB flexible Jonathan", Ph.D. dissertation, Université Joseph Fourier, Grenoble (France), 25th April 2002.

[15] Truyen E., Robben B., Vanhaute B., Coninx T., Joosen W., Verbaeten P., "Portable support for transparent thread migration in Java", *Proc. ASA/MA 2000*, Lecture Notes in Computer Science 1882, Springer, pp. 29-43.

[16] White J., "Telescript technology: the foundation for the electronic market place", *General Magic White Paper*, General Magic, 1994.

Web references

[17] Aglets - http://www.trl.ibm.com/aglets/index_e.htm

[18] ATHOS project (ITEA European program) - http://www.itea-athos.com/

[19] Foundation for Intelligent Physical Agents - http://www.fipa.org/

[20] Grasshopper - http://www.grasshopper.de/

[21] ObjectWeb Initiative - http://www.objectweb.org/

[22] OMG Agent Platform Special Interest Group - http://www.objs.com/agent/

[23] SOMA - http://lia.deis.unibo.it/Research/SOMA/