Moorea, a Service Execution Environment for Telecommunication Application

Bruno Dillenseger, Anne-Marie Tagant, Huan Tran-Viet, Laurent Hazard[†]
France Télécom R&D, DTL/ASR
28 chemin du Vieux Chêne, BP 98, 38243 Meylan cedex
[†]38/40 rue Général Leclerc, 92794 Issy Moulineaux cedex 9
email: {name (without "-")}.{surname (without "-")}@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 MASIF 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 explains how telecommunication service execution environments may benefit from these advanced features.

1. Introduction

In the context of advanced telecommunication services, we present a Java mobile agent platform that meets several critical requirements of service execution environments. As of today's research topics, involving (mobile and/or intelligent) agent technology in telecommunication infrastructure is a well-known trend (e.g. see a digest of several European projects in [INF 99]).

As far as mobility is concerned, application to optimization of processing and network resources consumption, as well as resistance to non-persistent network connectivity have been already put into light and experienced. Nevertheless, mobile agent technology often comes with several specific drawbacks, such as bad scalability, cost and complexity of agent activity transportation, unreliable agent communication and lack of interoperability between heterogeneous platforms. The specific design of our platform is motivated by these considerations.

Moorea is a Java mobile agent platform implementing the Mobile Agent System Interoperability Facilities specification [OMG 97]. 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 gives an overview of the upcoming application of Moorea in the field of telecommunication service execution environments.

2. Moorea Overview

2.1 Moorea Architecture

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

- a reactive object kernel (*Junior* [HAZ 99]) and a reactive language (*Rhum*, derived from the Esterel synchronous programming language [BER 92]) with Java environment and distribution support,
- with a Java mobile object framework (*SMI* [DIL 00]) implementing OMG's Mobile Agent System Interoperability Facilities specification [OMG 97] (also known as Mobile Agent Facilities),
- on top of a flexible Object Request Broker (*Jonathan* [OBJ 00]) offering both a Java-RMI and a CORBA personality (API);
- transparency to mobility support has been added to Jonathan, Jeremie and Rhum layers.

On the one hand, this Java, CORBA and MASIF based approach is relevant to hardware and operating system independence, as well as portability and interoperability enforcement at various levels (ORB architecture, standard IDL-defined interfaces, common conceptual framework). On the other hand, the mapping of the reactive object model [BOU 96] to mobile agents is suited to providing an efficient, reliable and scalable environment for service development and execution.

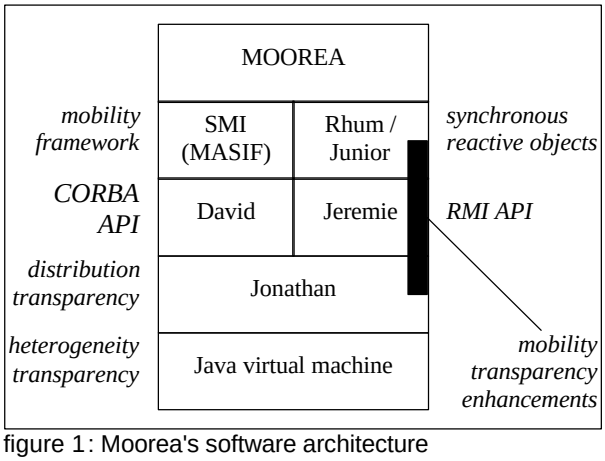

figure 1: Moorea's software architecture

2.2 Key Concepts

2.2.1 Reactive Agent, Agent Reference

Agent activity is fully defined by a reactive behavior specification in a dedicated programming language called Rhum (see section 2.2.3), and a passive Java object whose methods are invoked by the reactive behavior to perform basic computations. The reactive behavior is compiled into a Java reactive object, embedding the associated passive object (see figure 2).

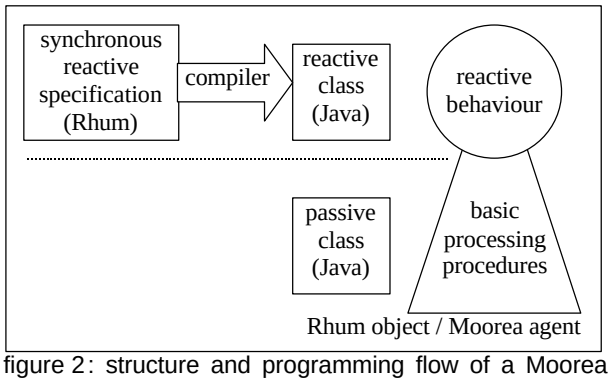

figure 2: structure and programming flow of a Moorea agent.

Agents are designated by their *reactive reference* (or *Moorea object reference*), which may be used for managing them or for sending them events, in a distributed way.

2.2.2 Synchronous Execution and Instants

Moorea's specific reactive model derives from synchronous programming (e.g. Esterel [BER 92]). In such a model, the execution time is sliced into logical instants, which define the lifetime of an event and the reaction semantics (figure 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 in 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.

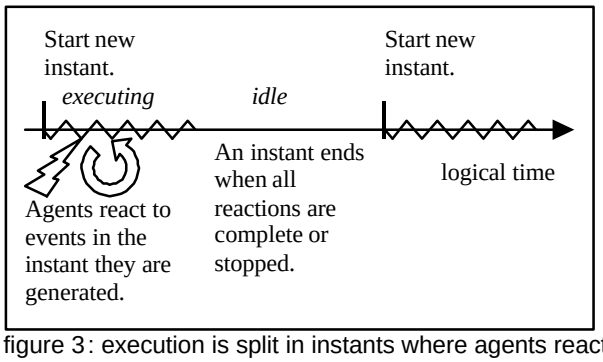

figure 3: execution is split in instants where agents react to and generate events.

2.2.3 Reactive Synchronous Language Rhum

A Rhum program defines parallel branches, synchronization, loops, event generation and event waiting (see example in figure 4). Derived from synchronous language Esterel [BER 92], Rhum modifies the semantics to avoid causality problems and to allow dynamic program composition [BOU 96].

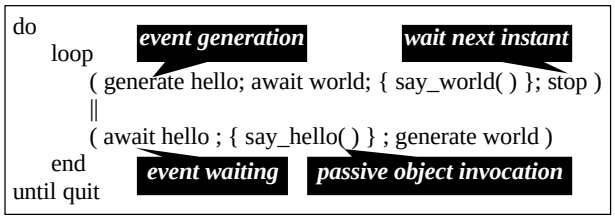

figure 4: this Rhum program defines a loop on two parallel branches, whose execution results in calling on the passive object method `say_hello()` first and then method `say_world()`. This order is enforced by ad hoc generation and waiting of events `hello` and `world`. Note that without the `stop` instruction, the program would endlessly loop in a unique, never-ending instant. Here, each loop takes one instant. The program exits at the end of an instant when event `quit` is present.

2.2.4 Agency, Reactive Domain

Agents are executed by, and move between *agencies*. Agencies define both their localization and the reactive domain they belong to. A *reactive domain* is an execution environment for reactive agents, which controls the sequence of instants, the event dispatching and reaction execution.

2.2.5 Events

Events are identified by a name. There are two kinds of event (figure 5):

- an environment event is generated for a whole reactive domain, and any agent waiting for such an event in this domain reacts;
- a targeted event is addressed to the reactive interface of a particular agent, designated by its reactive reference, located in any reactive domain (local or remote). Targeted events may hold arguments.

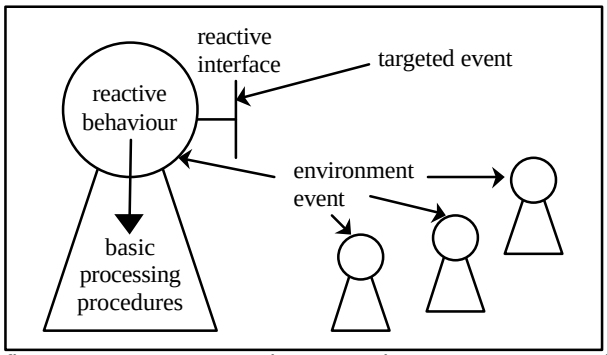

figure 5: an agent reacting to environment events and targeted events

2.2.6 Agent Lifecycle

Moorea agents implement a lifecycle interface so as to be aware of – and possibly react to – lifecycle stages. Stages are: after agent creation, before the agent moves, after an agent has successfully moved, after a move has failed, before the agent activity is suspended or resumed, before an agent is terminated, and before an agent's host agency is shut down.

All these stages are associated to dedicated callbacks on the agent. For example, an agent typically initializes itself after its creation, or may decide to move to another agency when its current host agency is being shut down. Some of these callbacks may throw an exception, which means that the corresponding stage is not acceptable to the agent. For example, an agent that cannot initialize correctly, or that is not able to move for some reason, should throw an exception, respectively in the "after creation" and the "before move" callback.

3. Technical Issues

3.1 Reactive Domains are not Distributed

Each agency is a reactive domain, and domains are not distributed: the instants sequence is specific to each domain, with no synchronization between domains. This choice springs from performance considerations: managing a distributed domain would require complex distributed algorithms, resulting in a communication overhead, and in having the slowest execution node ruling instant duration for all nodes. Moreover, a network or execution node hang-up would freeze the whole distributed reactive domain.

As a consequence, the synchronous property (i.e. events are present in the instant they are generated) only applies to events staying in the agency where they are generated (i.e. all environment events, and targeted events addressed to local agents). If an application relies on the synchronous property, then the involved agents should be located in the same agency; obviously, mobility makes it possible in a dynamic way.

3.2 Strong Mobility Support

In the mobile computation community, [FUG 98] defines 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.

3.2.1 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. The moving agent is transported with its behavior and execution state, and then resumed in the new reactive domain, in a new instant.

3.2.2 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 leads to a fragile and inefficient *reference chain* (i.e. a chain of several indirections), relying on every visited agency to keep running, which is not realistic, and against a major mobility justification;
- 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 – see section 2.1). 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.

3.3 Agent Management

Moorea offers two standardized management interfaces on CORBA, as specified by MASIF [OMG 97]. Agencies implement interface `MAFAgentSystem` to offer creation and termination of agents, reception of moving agents, suspension and resumption of agent activity, termination of the agency, and local agent listing. A "finder" service implements interface `MAFFinder`, for agent and agencies registration and lookup.

It appeared that a few useful operations were missing in MASIF interfaces, and we extended each of them with two extra operations, via inheritance. For example, interface `MAFAgentSystem` has been enriched with a `move_agent` operation, enabling to ask an agency to move an agent away (`MAFAgentSystem` only allows to ask an agency to receive an agent).

3.4 Interworking between Reactive and non-Reactive Software

3.4.1 Constraints on Instant and Reaction Duration

The synchronous programming model assumes that instants have a null duration, which means that the synchronous system reacts quicker than the environment it is bound to. Meeting this requirement requires very short reactions.

This constraint is taken into account in Moorea. For example, many remote management methods are performed with "future" semantics. This means that the call immediately returns a call identifier, and that an associated callback will later give the result or the exception of the actual call.

3.4.2 Java Method Invocation on a Reactive Agent

Reactive agents lifecycle is managed through a Rhum program, detecting lifecycle events and invoking associated lifecycle callbacks on the passive object. The difficulty springs from the fact that SMI kernel has to wait the callback completion, and to get the thrown exception, if any (see section 2.2.6), while events are one-way (asynchronous).

When SMI kernel invokes a lifecycle callback on an agent, it is translated into an event, handled by the reactive domain's thread, while the caller thread is blocked (see figure 6). Then, the reactive domain's thread invokes the actual callback on the passive object, and finally awakes the caller's thread. If the callback throws an exception, it is forwarded to the caller thread.

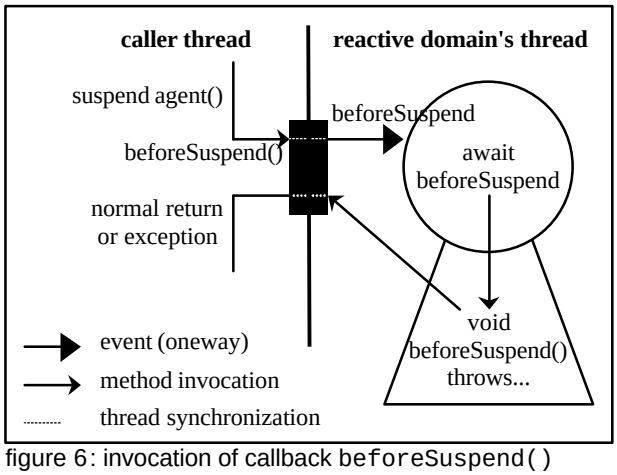

figure 6: invocation of callback `beforeSuspend()`

4. Related Work

4.1 Middleware Approach, Interoperability

While today's mobile agent platforms generally don't follow a middleware approach, not only Moorea is built on ORB Jonathan, but its mobility feature is fully CORBA-based and MASIF compliant. Moorea agents and agencies communicate through the ORB, using common associated services such as naming service. Previous similar approaches, and their benefits in terms of code reuse and agent interoperability, have already been described in [DIL 98]. IKV++'s Grasshopper is the only other known MASIF-compliant platform.

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

4.2 Transparency to Mobility

Most of mobile agent platforms take care of transporting the code and agent "static" state (i.e. classes and attributes values) 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 and General Magic's Telescript 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 [TRU 00]).

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 are clearly identified by the end of instants. After a move, an agent transparently resumes its activity in a new instant, at a new location. Moreover, this transparency support is

provided at low cost, by avoiding to transport "expensive" items like execution threads.

4.3 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 [BER 00].

The Bond agent system [BOL 00] 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 [ACO 01]. 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.

5. Application to Telecommunication Service Execution Environment

5.1 Requirements of Telecommunication Services

In the context of the ATHOS project, Moorea is going to be used to supply 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 networked 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.

On the one hand, *distribution* would be valuable for a number of reasons: first of all, 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 calls to these services simultaneously. 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.

On the other hand, it should be "*component*"-oriented to ease the decomposition of the overall complexity, the reuse of existing code, the dynamic distribution, while separating architecture definition, assembling and management from pure software programming. The *management* features for the service components should support deployment, upgrade, monitoring, troubleshooting, dynamic load-balancing...

Finally, the creation and execution environment should hide hardware and operating system *heterogeneity* of execution nodes, in order to limit service development overhead and avoid to be bound to one particular computing environment (hardware, operating system). Common service management and interoperability between services are also key issues.

5.2 Agent-Based Approach

Among the variety of existing component-oriented models, the agent paradigm retains service designers' attention. Agents may be regarded as components encapsulating data, processing capabilities and behavior. As a consequence, agents offer a high level of modularity and autonomy, which are key properties for decomposing services into autonomous – but interacting – components, that can be distributed and managed over a network. Besides the benefit of splitting complexity into small independent and reusable modules, implementing services as autonomous agents also allows a service to fail without disturbing the others, and makes it possible to selectively upgrade some parts of services, without necessarily shutting down any node nor any other service.

5.3 Reactive Agents

The execution model of reactive agents is based on reacting in sequence (i.e. not in parallel) to asynchronous messages ("events"). This model both saves processing time (no preemptive scheduling and no context switching) and avoids concurrency-related troubles (consistency issues, deadlocks between critical sections in mutual exclusion).

This model thus enforces good scalability and reliability properties, while simplifying the programmer's task regarding concurrent access and management of execution threads. Moreover, using a reactive language such as Rhum is convenient to implement communication protocol control.

5.4 Reactive Agents Mobility

Moorea's mobility capabilities make it possible to optimize utilization of computing and networking

resources. For example, it is possible that particular phases of a service execution require a lot of message exchange between two (or more) agents. In this case, it may be interesting to place these agents in the same execution node in order to save network bandwidth.

Moreover, it may be valuable to move an agent to another machine in various cases: because its current execution node is overloaded (load balancing), because a new node has just been added to the distributed execution environment (load balancing and system maintenance), or because its current execution node is going to be shut down (system maintenance).

Lastly, it is important to note that the combination of mobility and reactive model offers a clear mobility semantics, while considerably limiting mobility overhead (see section 3.2.1). This approach must be compared to thread-based mobile agent activity, whose mobility is either disturbing (the execution state is reset to some default state after move), or complex and costly (mobility includes execution stacks...), while introducing (sometimes unclear) side effects to the programming semantics. Once again, we see that the reactive model is a relevant approach to improve scalability and reliability.

5.5 Interoperability Concerns

Telecommunication services generally consist of vertically integrated platforms, that typically don't interoperate with each other, and come with their own management system.

To improve this situation, Moorea's architecture is based on middleware, including several standard layers and interfaces, namely CORBA, Java RMI and MASIF. Beyond this software infrastructure, and although agent interoperability is not directly addressed by Moorea, it has to be noted that Rhum would be particularly convenient to both specify and implement speech acts based protocols/dialogs as specified by FIPA's Agent Communication Language.

6. Conclusion

Moorea is a 100% Java mobile agent platform, based on standard software infrastructure (Java-RMI, CORBA, MAF/MASIF), 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.

Thanks to Rhum synchronous reactive language, Moorea offers the opportunity to conveniently implement telecommunication services by separating the service logic from the basic actions. More than convenience, the high level representation of a service logic helps troubleshooting, extendibility and simulation, and probably also provability of a few properties. Moreover, Moorea agents being highly mobile (high level of transparency, low-cost mobility), they are good candidates for dynamic load-balancing of executing services. Directions for future work on Moorea include performance evaluation, dynamic load-balancing, dynamic versioning, and provability.

Acknowledgement

Moorea is developed in the context of the ATHOS project of the ITEA European research program.

References

[ACO 01] ACOSTA BERMEJO, R., "Programming in REJO", to be published in *Réseaux et systèmes répartis - Calculateurs parallèles*, special issue "Evolutions dans le domaine des intergiciels", Hermes ed., 2001.

[BER 92] BERRY G., GONTHIER G., "The Esterel Synchronous Language: Design, Semantics, Implementation", *Science of Computer Programming*, 19(2), 1992.

[BER 00] BERTIN V., POIZE M., PULOU J., SIFAKIS J., "Towards Validated Real-Rime Software", *proc. 12th Euromicro Conference on Real-Time Systems*, Stockholm, june 2000.

[BOL 00] 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.

[BOU 96] BOUSSINOT F., DOUMENC G., STEFANI J.-B., "Reactive Objects", *rapport de recherche* INRIA No 2664, october 1995.

[DIL 98] 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.

[DIL 00] 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.

[FIP 98] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, "FIPA 98 Specification", 1998, http://www.fipa.org/.

[FUG 98] FUGETTA A., PICCO G.-P., VIGNA G., "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, vol. 24, No 5, 1998, pp.342-361.

[HAZ 99] HAZARD L., SUSINI J.-F., BOUSSINOT F., "The Junior reactive kernel", *Rapport de recherche Inria 3732*, 1999.

[INF 99] INFOWIN PROJECT, "Agents Technology in europe, ACTS Activities", 1999, ISBN 3-00-005267-4.

[OBJ 00] OBJECTWEB INITIATIVE, "Jonathan : a white paper", 6th october 2000, http://www.objectweb.org/.

[OMG 97] OBJECT MANAGEMENT GROUP, "Mobile Agent System Interoperability Facilities", *TC document orbos/97-10-05*, 1997. Revised in "Mobile Agent Facilies", *formal/2000-01-02*, 2000.

[TRU 00] 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.