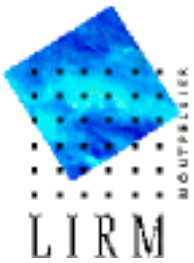

LABORATOIRE D'INFORMATIQUE, DE ROBOTIQUE
ET DE MICROÉLECTRONIQUE DE MONTPELLIER

Unité Mixte CNRS - Université Montpellier II C 09928

RAPPORT DE RECHERCHE

The MADKIT Agent Platform Architecture

Olivier GUTKNECHT
gutkneco@lirmm.fr
Jacques FERBER
ferber@lirmm.fr
Fabien MICHEL
fmichel@lirmm.fr

05/2000 R.R.LIRMM 000xx

161, Rue Ada - 34392 Montpellier Cedex 5 - France
Tél: (33) 67 41 85 85 - Fax: (33) 67 41 85 00

Résumé

In this report, we present MadKit (multi-agent development kit), a generic multi-agent platform. This toolkit is based on a organizational model. It uses concepts of groups and roles for agents to manage different agent models and multi-agent systems at the same time, while keeping a global structure.

We discuss the architecture of MadKit, based on a minimalist agent kernel decoupled from specific agency models. Basic services like distributed message passing, migration or monitoring are provided by platform agents for maximal flexibility. The componential interface model allows variations in platform appearance and classes of usage.

We illustrate this approach by explaining some consequences of the architecture and describe our motivation for MadKit design : integration of heterogeneous agent applications, reuse of agent social structure patterns, and versatility of the application platform. A summary is given of some MadKit-based applications to date.

1 Introduction

1.1 The heterogeneity issue

A major characteristic in agent research and applications is the high heterogeneity of the field.

By heterogeneity, we mean both *agent model heterogeneity*, characterizing agents built and described with different models and formalisms ; *language heterogeneity*, with agents using different communication and interaction schemes, and finally *applicative heterogeneity*, as multi-agent systems are used with various goals and in many applicative domains.

Many successful theories and applications has been proposed in different fields of multi-agent research : interface agents [8], mobile agents [7], information retrieval agents [11], etc.

We believe that being able to take advantage of this diversity of approaches simultaneously is important to build complex systems while keeping heterogeneity manageable. An “one size fits all” seems rather adventurous ; thus, the interesting question is how to establish conceptual models *and* software toolkits to facilitate integration.

We also advocates that interoperability in agent system should be envisaged at agent level. Existing interoperability mechanisms in software engineering (CORBA, XML, ...) are interesting for the foundations they procure, but are not the universal answer to our preoccupation. At least, the relationship between the underlying interoperability platform and the agent layer be clearly defined and identified.

1.2 MadKit as a multi multi-agent system

We took the agent platform field as an example, but the same diversity of approaches exists within agent formalisms, communication models, and architectures.

We have designed a model, called AALAADIN, that structures multi-agent systems, and implemented a platform based on this model. The platform itself has been realized to take full advantage of the model. In this report, we will particularly focus on the MADKIT platform. We will see that a structural model at multi-agent systems level can ease agent diversity integration within a platform, hence this qualification of “multi multi-agent systems”.

The MADKIT toolkit was motivated by the need to provide a generic, highly customizable and scalable agent platform. The goal of building a foundation layer for various agent models was essential, as well as making the basic services provided completely extensible and replaceable.

We briefly introduce the AALAADIN conceptual model in section 2. Section 3 describes the platform architecture. It presents the concept of "agent micro-kernel", how system services are provided by agents and gives an overview of the agent interface model. Section 4 presents some experiments and systems built with MADKIT. Section 5 briefly talks about future work and concludes.

2 The agent/group/role model

The MADKIT platform architecture is rooted in the AGR (agent-group-role) model developed in the context of the AALAADIN project. MADKIT both implements and uses for its own management this model. We will just summarize it here, and refer to [3] for a more general overview of the project and [4] for a detailed description of its formal operational semantics.

We advocate that considering organizational concepts, such as groups, roles, structures, dependencies, etc. as first class citizens might be a key issue for building large scale, heterogeneous systems.

In this definition, an organization is viewed as a framework for activity and interaction through the definition of groups, roles and their relationships. But, by avoiding an agent-oriented viewpoint, an organization is regarded as a structural relationship between a collection of agents. Thus, an organization can be described solely on the basis of its structure, i.e. by the way groups and roles are arranged to form a whole, without being concerned with the way agents actually behave, and multi-agent systems will be analyzed from the "outside", as a set of interaction modes. The specific architecture of agents is purposely not addressed.

The AALAADIN model is based on three core concepts : *agent*, *group* and *role*. Figure 1 presents a diagram of this model.

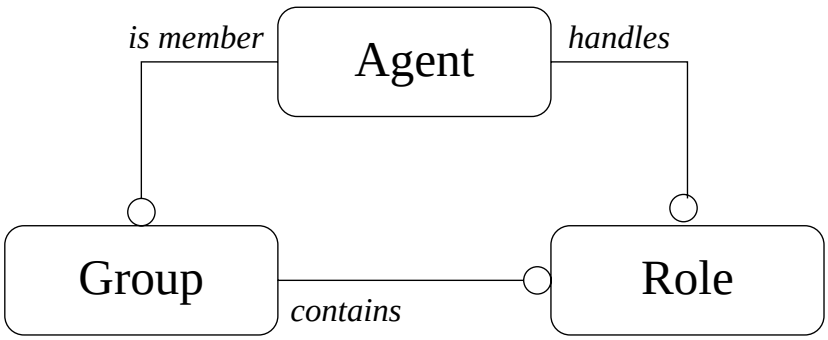

FIG. 1 – The core model

2.1 Agent

The model places no constraints on the internal architecture of agents. An *agent* is only specified as an active communicating entity which plays *roles* within *groups*. This agent definition is intentionally general to allow agent designers to adopt the most accurate definition of agent-hood relative to their application. The agent designer is responsible for choosing the most appropriate agent model as internal architecture.

2.2 Group

Groups are defined as atomic sets of agent aggregation. Each agent is part of one or more groups. In its most basic form, the group is only a way to tag a set of agents. In a more developed form,in conjunction with the role definition, it may represent any usual multi-agent system. An agent can be a member of n groups at the same time. A major point of AALAADIN groups is that they can freely overlap. A group can be founded by any agent.

2.3 Role

The role is an abstract representation of an agent function, service or identification within a group. Each agent can handle multiple roles, and each role handled by an agent is local to a group. Handling a role in a group must be requested by the candidate agent, and is not necessarily awarded. Abstract communication schemes are thus defined from roles.

The model is not a static description of an agent organization. It also allows to define rules to specify the part of the dynamics of the agent organization. Note that the particular mechanism for role access within a group is not defined (systematic acceptance or refusal, admission conditioned by skills or by an admission dialog, relation to a group metrics,...).

2.4 Discussion

We see overlapping groups as a trade-off between a flat world were agents are not part of any structure and models in which organizations of agents are considered as atomic agents, which raises important problems of coherence, structure and semantics.

In our model, when agentification of a group seems necessary to the designed , a role of “representative” of the group is associated to one of its member to act as the proxy for the whole group for the external world.

In a real-world metaphor, this could be compared to the situation when a person is in negotiation with a company. The company as a whole is never part of the interaction : instead, an individual is the *representative* of the company in the dialog.

3 Architecture

The MADKIT platform is built around this model. In addition to the three core concepts, the platform adds three design principles :

- Micro-kernel architecture
- Agentification of services
- Graphic component model

MadKit itself is a set of packages of Java classes that implements the agent kernel, the various libraries of messages, probes and agents. It also includes a graphical development environment and standard agent models

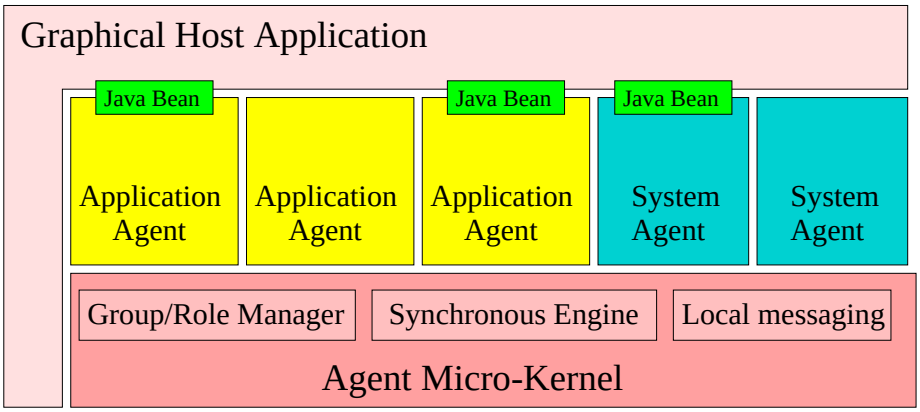

FIG. 2 – MadKit Architecture Diagram

The basic philosophy of the MADKIT architecture is to use wherever possible the platform for its own management : any service beside those assured by the micro-kernel are handled by agents. Thus the platform is not an agent platform is the classical sense. The reduced size of the micro-kernel, combined with the principle of modular services managed by agents enable a range of multiple, scalable platforms and construction of libraries of specialized agent models.

Agent groups have been proposed in other architectures, such as [1], although the mechanism is specific to mobile agents and lacks our ability to handle multiple groups and multiple roles in a generic model.

3.1 Agent micro-kernel

The MADKIT micro-kernel is a small (less than 40 Kb) and optimized agent kernel.

The term “micro-kernel” is intentionnaly used as a reference to the role of micro-kernels in the domain of OS engineering [9]. We could directly translate their motto into : *‘incorporating a number of key facilities that allow the efficient deployment of agent toolkits.”*

The MADKIT kernel only handles the following tasks :

Control of local groups and roles As most of the interoperability and extensibility possibilities in MADKIT relies on the organizational layer, it is mandatory that group and role are handled at the lowest level in the platform, to provide this functionality to any agent The micro-kernel is responsible for maintaining correct information about group members and roles handled. It also checks if requests made on groups and roles structures are correct (ie : evaluating - or delegating evaluation - of role functions).

Agent life-cycle management The kernel also launches (and eventually kills) agents, and maintain tables and references of agent instances, it is the only module in MADKIT that owns direct references to the actual agents. It also handles the

agent personal information and assigns it a globally unique identifier (kernel address plus agent identification on the local kernel) , the `AgentAddress` upon creation. This identifier can be redefined to accept standardized agent naming schemes.

Local message passing The kernel manages routing and distribution of messages between **local** agents. The basic mechanism relies on a copy-on-write implementation to avoid unnecessary operations.

The kernel itself is wrapped in an special agent, the `KernelAgent`, which is created at bootstrap. It permits control and monitoring of the kernel within the agent model.

Kernel hooks

The kernel itself is fully extensible through "kernel hooks". Any entitled agent (i.e. an agent that has been allowed to be member of the *system* group) can request to the `KernelAgent` to subscribe to a kernel hooks.

These hooks are the generic subscribe-and-publish scheme allowing extension of the core behavior of the platform. Every core function in the kernel (adding an agent to a group, launching an agent, sending a message) implements this mechanisms.

Monitor hooks Any number of agents can subscribe to a monitoring hook. In this type of hook, invocation of a kernel operation make the `KernelAgent` send an "inform" message to the agents that requested a monitor hook on this operation, with the arguments of this operation as the content of the message. For instance, this is how are written the agents that monitor the population or organization in the platform.

Interceptor hooks These hooks are similar to the previous type, but only one agent can hold an interceptor hook on a give kernel operation. Furthermore, an interceptor hook prevents the execution of the implemented kernel operation. It is particularly useful to write distributed messaging agents, group synchronizers, security control in groups, etc. by changing the behavior of a basic operation. The reduced number of kernel calls and the simple underlying model helps the modification of these call while preserving their semantics.

When a kernel call having associated hooks is invoked, the information is transmitted to the `KernelAgent` which does the message transmission.

Kernel Operation

Hooks enable monitoring and change of behavior, but the *system* group defines additional interactions between a member and the `KernelAgent` to allow *actions* on the kernel.

For instance, a system *communicator* agent can inject in the local kernel a message received through a socket connection with a distant madkit platform.

3.2 Agents

The generic agent is MADKIT is a class defining basic life-cycle (what to do upon activation, execution, and termination).

– Control and life-cycle. The main agent class in MADKIT defines primitives related to message passing, plus group and role management, but does not implement a specific execution policy. A subclass adds support for concurrent, thread-based execution, which is the natural model for coarse-grained collaborative or cognitive agent. Additional subclasses implements synchronous execution through an external scheduler, focused on reactive or hybrid architectures : many fine-grained agents.
– Communication is achieved through asynchronous message passing, with primitives to directly send a message to another agent represented by its `AgentAddress`, or the higher-level version that send or broadcast to one or all agents having a given role in a specific group.
– Group and roles actions and requests are defined at action level. The agent developer is completely free to define the agent behavior, but the organizational view will be always present.

Message passing

Messages in the MADKIT platform are defined by inheritance from a generic `Message` class. Thus specific messages can be defined for intra-group communication, and allows a group to have its specific communication attributes. Messages receivers and senders are identified with their `AgentAddress`. MadKit do not define interaction mechanism, which can be defined on an ad-hoc basis, or built in a specific agent model library.

Agent models

Several specific agent libraries have been built above this infrastructure, notably :
– Bindings to the Scheme language [2]
– An agent model that wraps the JESS rule engine [5].
– Various models for artificial life / reactive systems, such as a “turtle kit” that mimics some functions of the StarLogo environement [10].
– An actor model implementation
– Agent construction tools running themselves as agents in the platform 3.

3.3 Agentification of services

In contrast to other architectures, MADKIT uses agents to achieve things like distributed message passing, migration control, dynamic security, and other aspect of system management. These different services are represented in the platform as roles in some specific groups, defined in an abstract organizational structure. This allows a very high level of customization, as these service agents can be replaced without hurdle.

For instance, external developers have built a new *group synchronizer* agent that use a shared JNDI directory to maintain group information across distributed kernel instead of relying on our provided (but MadKit-specific) system. Their agent uses some hooks in the kernel to achieve its goal and replaced the provided *group synchronizer* agent by requesting the same role in the *system* group. Other agents did not notice the change.

The role delegation principle has the other interesting effect to allow easy scaling. An agent can hold several roles at the beginning of a group, and as the group grows, launches other agents and delegates them some of these roles.

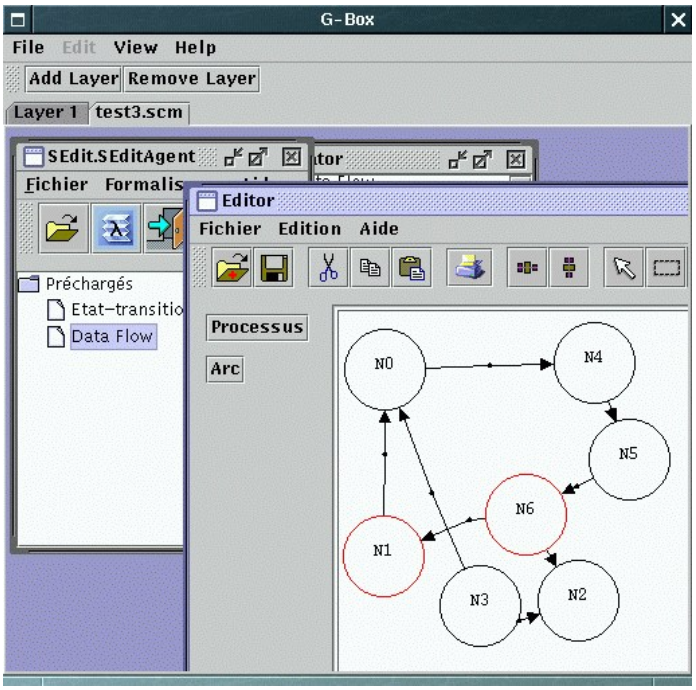

FIG. 3 – SEDIT tools running as agents in MADKIT

Communication and distribution

As messaging, as well as groups and roles management use the `AgentAddress` identifier, and as this identifier is unique across distant kernels, MADKIT agents can be transparently distributed without changing anything in the agent code. Groups can spawn across different kernels, and agents usually do not notice it.

Distribution in the agent platform relies on two roles in the *system* group :

- The *communicator* agent is used by the micro-kernel to route non-local messages to other *communicator* agents, on distant platforms, which therefore injects the now-local messages in their kernel (see figure 4)
- The *group synchronizer* agents allows groups and roles to be distributed among kernels by sending groups and roles changes to other synchronizers, which in turn enter these information in their local kernel. Note that these group synchronizers use themselves their own distributed group to ease distributed group management.

Since the communication mechanisms are built as regular agents in the platform, communication and migration could be tailored to specific platform requirements only by changing the communication agents, for instance in disconnected mode for laptop.

An MADKIT platform can run in full local mode just by not launching the communication agents.

These services are not necessarily handled by only one agent. For instance the *communicator* agent can be the *representative* for a group gathering agents specialized in sockets or CORBA communications and delegate the message to the appropriate agent

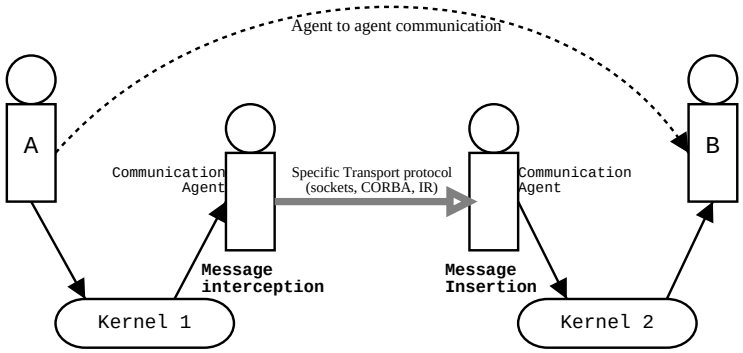

FIG. 4 – Communication agents

3.4 Componential graphical architecture

MADKIT graphic model is based on independent graphic components, using the Java Beans specification in the standard version.

Each agent is solely responsible for its own graphical interface, in every aspects (rendering, event processing, actions...) An agent interface can be a simple label, a complex construction based on multiple widgets, or a legacy bean software module. A “graphic shell” launches the kernel and setup the interfaces for the various agents and manage them in a global GUI (for instance : each agent has its own window, or is mapped in a global worksheet, or is combined with another agent interface,...).

As the graphic shell is a classic software module, it can be wrapped in an agent for maximum flexibility, allowing control of other agent interfaces by a regular MADKIT agent that can be part of any interaction scenario.

3.5 Consequences and discussion

The conjunction of the agent micro-kernel and the decoupled agent GUIs as well as a modular set of agent services allows important customizations of the MADKIT platform (figure 5). For instance, the following “varieties” of the platform have been developed :

- A complete graphical environment to develop, and test multi-agent systems, called the “G-Box”. It allows graphical control of agent life-cycle (launch, termination, pause), dynamic loading of agents, and uses introspection on agent code to discover at runtime groups and roles, references to other agents, and offer direct manipulation of these structures.
- A text-only mode, only running the micro-kernel without instantiating graphical interfaces of running agents. This platform is useful to keep a small “agent daemon” running on a machine and agents providing services of brokering, naming or routing for other machines.
- An applet wrapper, which carries the agent micro-kernel with some application agents, and executes in a distant browser.
- A “classic” application that would embed a MadKit kernel and hosts agents that handles the collaborative / dynamic aspects. For instance, this was the imple-

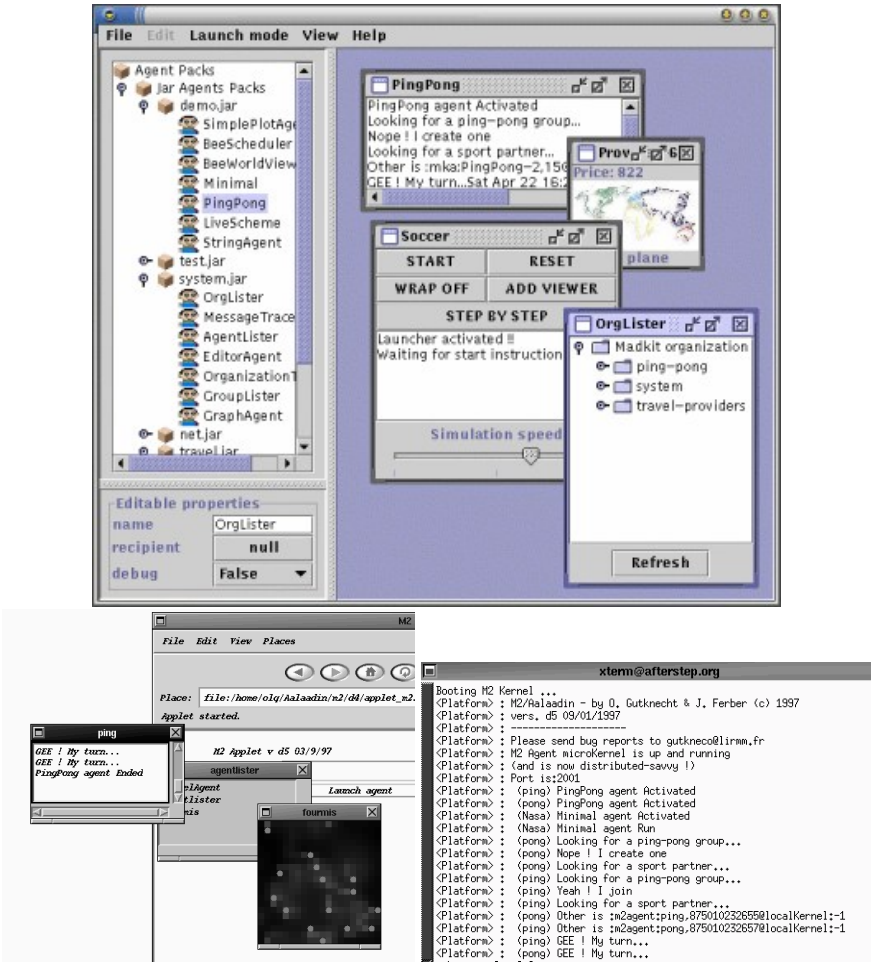

FIG. 5 – MADKIT running in applet, G-Box and console modes

FIG. 6 – A simplistic example of a MadKit kernel and two agents on a Palm device

mentation choice made in the Wex application described below.

– Something more experimental is a version of MADKIT 3.5 tailored for the Palm Pilot. The kernel is slightly tweaked to only use the set of classes allowed on the Java Platform Micro Edition[6]. A specific *communicator* handles infrared messaging.

4 Applications

MadKit has been used in various research teams for nearly two year in projects covering a wide range of applications, from simulation of hybrid architectures for control of submarine robots to evaluation of social networks or study of multi-agent control in a production line.

For instance, Wex, developed by Euriware S.A., is a complex MADKIT application for knowledge-management applications. It federates information from different data sources (databases, support tools, web search engines, current page browsed by the user and parsed...) and present unified views of these highly heterogeneous knowledge sources. Users can maintain shared ontologies on their domain. Agents have been implemented to encapsulate the various mechanisms to retrieve and transform information. The abstract organizational structure has been defined, and the various agents can plugged in to adapt the platform to the client specific needs.

5 Conclusion and future work

In this report, we presented an agent toolkit based on an organizational model, and we argued that that large and complex agent systems should be able to cope with heterogeneity of models, communications and individual agent architectures.

We plan to extend this work in three directions. We plan to continue work on the underlying model, especially in the context of formal expression and design methodologies. Secondly, we will extend the platform itself by refining existing system agents, proposing more predefined agents and groups libraries. Finally, we are planning to built additional layers and models for multi-agent based simulation.

Références

[1] Joachim Baumann and Nikolaos Radouniklis. Agent groups in mobile agent systems. In *IFIP WG 6.1, International Conference on Distributed Applications and Interoperable Systems (DAIS 97)*, 1997.

[2] Per Bothner. Functional scripting languages for the jvm. In *3rd Annual European Conference on Java and Object Orientation*, Arhus, Denmark, 1999.

[3] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Third International Conference on Multi-Agent Systems (ICMAS '98) Proceedings*, pages 128–135. IEEE Computer Society, 1998.

[4] Jacques Ferber and Olivier Gutknecht. Operational semantics of a role-based agent architecture. In *Proceedings of the 6th Int. Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1999.

[5] Ernest J. Friedman-Hill. *Jess, The Java Expert System Shell*. Distributed Computing Systems, Sandia National Laboratories, Livermore, CA, 2000.

[6] Sun Microsystems Inc. The k virtual machine white paper. Technical report, 2550, Garcia Avenue, Mountain View CA 94043, 1997.

[7] Frederick C. Knabe. An overview of mobile agent programming. In *Proceedings of the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*, Stockholm, Sweden, June 1996.

[8] Brenda Laurel. Interface agents : Metaphors with character. In Brenda Laurel, editor, *The Art of Human Computer Interface Design*, pages 355–365. Addison-Wesley, 1990.

[9] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, and Richard Sanzi. Mach : A foundation for Open Systems. In *Proceedings of the 34th Computer Society Ithe Second Workshop on Workstation Operating Systems(WWOS2)*, September 1989.

[10] Mitchel Resnick. *Turtles, Termites, and Traffic Jams : Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[11] E. M. Voorhees. Software agents for information retrieval. In O. Etzioni, editor, *Software Agents — Papers from the 1994 Spring Symposium (Technical Report SS–94–03)*, pages 126–129, March 1994.