Towards a Reference Model for Surveying Mobile Agent Systems

ALBERTO RODRIGUES DA SILVA alberto.silva@acm.org
IST / INESC, Rua Alves Redol, 1, 1000 Lisboa, Portugal
ARTUR ROMÃO artur@dmat.uevora.pt
University of Évora, 7000 Évora, Portugal
DWIGHT DEUGO deugo@scs.carleton.ca
Carleton University, Ottawa, Canada
MIGUEL MIRA DA SILVA mira-da-silva@ip.pt
IST / INESC, Rua Alves Redol, 1, 1000 Lisboa, Portugal

Abstract. There are increasing numbers of systems and research projects involving software agents and mobile agents. However, there is no reference model or conceptual framework to compare the resulting systems. In this paper, we propose a reference model to identify, classify and evaluate mobile agent systems having a significant set of non-trivial architectural issues and technical and functional features in order to support agent-based applications. Our proposed reference model describes a generic and global architecture and identifies a set of technical and functional features. The items analyzed include the following: execution, management of agent types, management of identifiers, persistence, navigation, communication, interaction with external resources, and security.

We apply this reference model to analyze, compare, and discuss some well-known mobile agent systems: Telescript, Aglets, ffMain, D'Agents, and AgentSpace.

Keywords: mobile agent systems, mobile agents, agent-based applications

1. Introduction

There are many mobile agent system proposals these days – such as Aglets Workbench [IBM97, LO98], D'Agents [Gra95, Dar98], Grasshopper [IKV98], Odyssey [GM97], Tacoma [JRS95], Concordia [WPW+97, ME99], Jumping Beans [AdA99], AgentSpace [SMD98], Mole [SBH96], MOA [MFC98], and PageSpace [CKTV96] – each with similar features and functionality. Nevertheless, they contain important technical and even conceptual differences, which are extremely important to understand.

Telescript [Whi94, Whi97] was the standard reference for mobile agent (MA) systems in 1995 because it was innovative and it was a breakthrough in agents. However, Telescript failed because it was huge, unstable, had poor performance, and included a difficult-to-learn programming language. On the other hand, ffMAIN [LDD95] was language and system independent. However, it has severe limitations affecting overall performance and is difficult to use for developing complex applications. The Aglets Workbench – today the reference Java-based MA system – lacks an elaborate object model, e.g., a notion of hierarchically organized execution places or management of agent families and clusters. It also lacks some technical capabilities, e.g., just two access control levels, no agent class manager, no "open channel" capability [AF89, Wel90] and no support for users transparently associated to agents.

We expect that in the future these (or other) MA systems will improve in order to support the development and execution of flexible, reliable, secure and efficient agent-based applications (ABAs). In order to achieve that objective, future MA systems should incorporate a combination of the best features from existing MA systems. For example, each should provide the following features:

- the protocol independence of TCP/IP and/or HTTP-based MA systems, as found in ffMain, in order to enable migration and agent communication independently of MA systems or specific programming languages;
- the "open channel" capability, as found in AgentSpace, in order to facilitate the task of ABA developers;
- the tight integration with Java, as supported by the Aglets Workbench, due to the fact that Java is the standard for developing mobile agent applications.

However, in spite of the increasing number of MA systems and the need to compare them, there are not, until now, any proposals for a standard MA system reference model.

We propose a first draft towards the construction of this model. The main benefits of having this reference model include the following:

- a common context to discuss agents, agent systems and agent-based applications;

- a better understanding of the components and functions associated with MA system architectures;
- a framework to compare, evaluate and benchmark current and future MA systems.

Others have worked on similar efforts with notable differences:

- The MASIF proposal [Mil98]. The MASIF specification contains a detailed conceptual model that was agreed upon by five MA system providers. Nevertheless, its focus is on the interoperability of MA systems through the adoption of two main interfaces: MAFAgentSystem, for agent transfer and management; and MAFFinder, for naming and locating. Furthermore, the MASIF specification was designed mainly for the OMG, consequently it is tightly coupled to the CORBA architecture.
- Electronic Catalogues. Web-based catalogues now provide a list of references for the majority of the most popular and new MA systems. In the appendix of this paper, we reference some of these catalogues. Of special note are the recent efforts from the University of Stuttgart with the Mobile Agent List (MAL) effort [MAL99], and the AgentBuilder's list of agent tools [ACT99]. In particular, MAL has the advantage that it allows MA system providers to add and modify their respective information following a pre-defined profile.

The paper is organized into 8 sections. In Section 2, we analyze mobile agent systems in the more general area of distributed agent-based applications and present related work. Section 3 introduces the main concepts and terminology adopted in this paper. Section 4 presents a generic and global architecture for agent systems supporting, developing and managing agent-based applications; an architecture that will be used to discuss the proposed reference model. Section 5 presents the core of the paper: it specifies and discusses the set of technical and functional features that are (or should be) provided by a MA system. Section 6 then presents a survey of existing MA systems and Section 7 discusses and compares all these systems according to the proposed reference model. Finally, Section 8 summarizes the main contributes of the paper. In the appendix, we present a significant set of electronic references related to mobile and software agents resources organized around the following topics: organizations, introductory readings, agent catalogues, specifications, and mobile agent systems and projects.

2. Scope and Related Work

It is critical to state the focus of this current work and to define its respective scope due to the fact that the subject of software agents, or in particular of mobile agents, is a relatively new, multidisciplinary area [Rie94, Nwa96, JSW98, Sil99].

One of the most usable agent definition is by Wooldridge and Jennings [WJ95], which discusses agents based on two basic notions: a weak and a strong definition of agents. The strong definition involves Artificial Intelligence (AI) techniques and models to characterize agents using mentalistics notions, such as knowledge, beliefs, intentions and obligations, or even emotional attributes. Based on their tentative agent definition, *the software agent perspective taken into account in this paper focuses on the weak definition for which autonomy, sociability, reactivity and mobility are the most important characteristics.*

A discussion and analysis of the interest and applicability of mobile agents is not the focus of this paper. We refer the reader to[CHK96, LO98, MDW99] and other introductory readings noted in the appendix for information on these topics.

Additionally, it is important to state that the focus of this paper is oriented more to MA system infrastructure than application. This means that the focus is based on low-level technical issues such as synchronization, thread management, communication, security and persistence. On the other hand, the focus of application-oriented approaches, such as those found in the AI field, is mostly on knowledge representation, cooperation/collaboration definition models, and agent communicationhigh-level.

Another aspect that should be clarified before we start is the relationship between MA systems and middleware systems, such as CORBA implementations, RMI, and DCOM. These latter types of systems present some similarities because they support the development and execution of distributed applications. However, MA systems provide a framework to develop applications based mainly on the agent paradigm. Whereas middleware systems use the object-oriented paradigm. For a detailed discussion concerning the analysis and relationship between these two paradigms the reader should consult [JSW98]. The main difference between these types of systems is better discussed along two dimensions: flexibility and specificity, as suggested by the Figure 1. MA systems are designed and implemented to support a well-defined types of agent-based applications, namely the emergent generation of (mobile) agent applications in the e-business context. On the other hand, middleware systems (such as DCOM, RMI or CORBA implementations) are more flexible because they offer low-level support features.

We expect the next-generation of MA systems to be developed on top of some kind of middleware system because it can provide several technical features (such as communication, security or persistence) that can be used directly and more easily by MA system architects. Figure 1 shows the relationships between the different classes of frameworks (or systems) involved with the support and development of distributed agent-based applications. We identify three inter-related classes of frameworks, each using/importing features provided by the lower one.

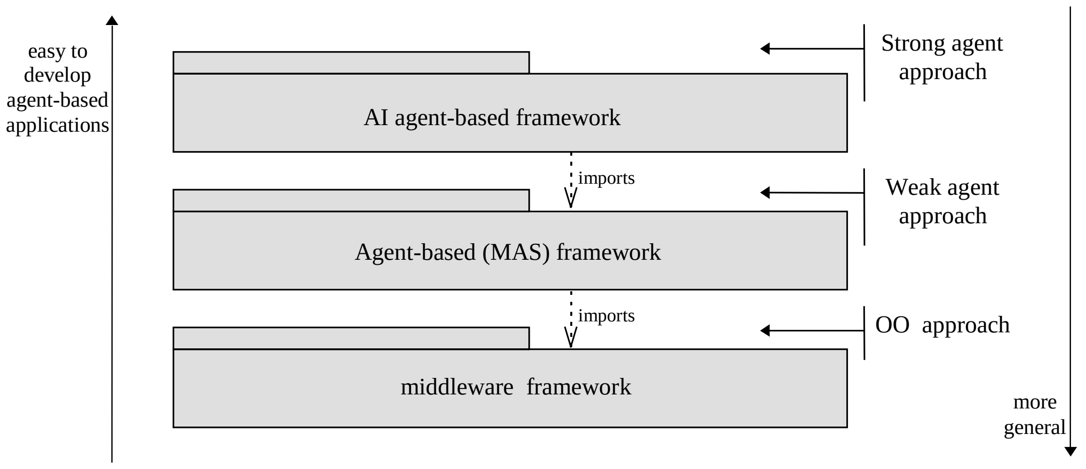


Figure 1: Frameworks to support agent-based applications

Middleware frameworks are positioned at the bottom level. They provide powerful and generic features based on the object-oriented approach, such as persistence, a name service, communication, location transparency, and mobility. Examples of this class of framework include DCE [Loc94], Iona's OrbixWeb [Ion98], Inprise's VisiBroker [Inp98], Sun's RMI [Sun97], ObjectSpace's Voyager [Obj99] and Microsoft DCOM [EE98].

At the middle level, agent-based frameworks can be defined based on the services provided by the middleware frameworks. These mid-level frameworks are less general than lower-level middleware frameworks, but provide a well-integrated set of components in order to facilitate the development of agent-based applications following the "weak agent" vision [WJ95].

Lastly, on top of the other frameworks, there are what can be called "agent-based applicational" frameworks. This kind of framework provides even more specific application components such as agent specializations for supporting well-defined tasks. One possible approach to these frameworks is based on AI, including support for knowledge representation, high-level communication protocols/languages, and intelligent agent-based models such as the BDI model [RG95]. RMIT [Ken+97] and Plangent [Ohs+97] are examples of these kind of applicational frameworks.

The main issues faced by frameworks based on the weak agent definition are discussed in this paper.

3. Concepts and Terminology

This section gives an overview of the main concepts and terminology for mobile agent systems needed in the rest of the paper and, in particular, to present the proposed reference model.

3.1 Agents, Agent Types, Agent Closures, and Nodes

An *agent* is the basic entity of the reference model. An agent executes specific tasks on behalf of someone (a person) or something (an organization or another agent) with some autonomy. An agent is an active object (i.e., it can decide to execute a method) that can be implemented through one or more threads on one operating system process.

An agent is an instance of some type, called in general *agent type* or *agent class*. An agent type can, due to organization and code distribution goals, be aggregated with other files, such as code, configuration, or image files. We call *agent closure* the set of all these files. So, to execute an agent it is necessary to have all the information defined by its closure. (The definition of closure introduced corresponds to a super set of the traditional notion of "code closure".)

A *node* is a hardware infrastructure on which agents are executed. Typically, a node is a computer, but in the near future it will probably include mobile/portable computers, e.g., personal digital assistants, personal intelligent communicators, hand-held PCs, TV sets, mobile phones, and other computing devices. Usually, an agent executes in one node. However, during its life cycle, it may navigate through an arbitrary number of nodes. An agent can be duplicated but each agent can only be executed in one node at a time.

3.2 Mobile Agent Systems

The system that is responsible for supporting and managing agents is called a *Mobile Agent* (MA) *System*. One or more MA systems can exist in each node. Sometimes a MA system is also called an "agent system". However, we prefer MA system because "agent system" is too general and sometimes vague or prone to misunderstanding. Other authors use different names for MA systems, such as "engine", "agent server", or "agent meeting point".

The MA system can be built from scratch or, more likely, as a combination of existing hardware infrastructures (e.g., ordinary PCs), operating systems (e.g., Windows or JavaOS), communications packages (e.g., TCP/IP, HTTP or CORBA), and some kind of compiler and/or virtual machine (e.g., Java VM).

The MA system provides a full computational environment to execute an agent, as well as other support features such as agent persistence, security and mobility. In order to support distributed applications and in particular agent mobility, different MA systems should communicate amongst themselves using a low-level protocols and agree on common agent representation formats (e.g., using well-known marshaling techniques as found in RPC and CORBA systems). The MA system should also provide (or at least integrate) specific APIs to allow access to external services and resources, such as databases (e.g., ODBC or JDBC), the file system and physical devices (mouse, screen, keyboard, or smart cards).

Context and Execution Places

The *context* is the public interface provided by each MA system. There is a direct and unique mapping between the MA system and the context, which means that each MA system is represented by a corresponding context.

The context can be organized in a set of logical entities designated by *execution places* or simply by *places*. This means that one context contains several places. Places are logical locations where agents execute, meet and communicate with other agents. Place and context are uniquely identified by an electronic address.

3.3 Agent-based Applications

An *agent-based application* (ABA) can consist of just one agent (e.g., a search information agent) but usually involves several agents that interact and communicate between themselves.

The typical metaphor used to talk informally about an ABA is the *community* (in this case, a community of agents). So, in this paper we will use both terms depending if the sentence is formal or informal.

A *community* (that is, an agent-based application) is formed by a set of agents that know how to behave, share their knowledge, and communicate using a common language.

Knowledge is a description of some fact, some relationship between facts and/or other relationships in some restricted contexts. Any agent in the same community can use the knowledge maintained by any other agent. It is likely that at the implementation level the knowledge of all agents belonging to the same community will be maintained by a database.

A community is a logical concept that can be spread over a number of nodes. An agent belongs to one or several communities if it is allowed to belong to those communities, can speak their respective languages and share their knowledge. There will be *guest agents* that are allowed only limited access to the community knowledge, probably introduced by another agent that is responsible for its behavior. Other agents can be specialized as *translators* for different languages, *mediators* to resolve conflicts or *police agents* to stop or kill agents with bad behavior.

A *homogeneous community* is a set of agents sharing the same knowledge and is supported by a common MA system. However, a set of agents supported by the same MA system does not define its own a community.

The notion of homogeneous community raises two new aspects. The first involves the need for a *communication language* between homogeneous agents. Basically there are two approaches: declarative (e.g., KQML [LF97]) or procedural and/or object-oriented (e.g., Tcl, Telescript, and Java). The second aspect involves how to represent the specific *knowledge* of the community. It also has two basic approaches: knowledge representation languages (e.g., KIF [GK94], XML specific DTD [W3C98]) and specific APIs and protocols agreed amongst the principal entities responsible for the development and management of the involved community. (These two aspects require a detailed discussion, which is not in the scope of this paper. The interested reader may consult [Rie94, Sin98] for more information).

A *heterogeneous community* is an extension of the community concept in terms of desired capabilities and complexity because it is a set of agents sharing the same knowledge, but supported by different MA systems. This class of agent-based applications requires interaction and communication between a heterogeneous set of agents.

For instance, in the current state-of-the-art MA systems, a Telescript agent and an Aglets agent cannot communicate directly using their standard primitives. Obviously, they can always interact using low-level communication mechanisms such as sockets or shared files. However, if they proceed using these mechanisms they cannot take advantage from their high-level primitives integrated with their respective object models and, for instance, their security features.

In this case agent communication should be independent of any language or MA system. In this scope there are two main efforts not necessarily incompatible: (1) the KSE effort with the KQML language [LF97]; and (2) the FIPA's ACL [FIPA97].

Figure 2 shows the relationships between an agent and homogeneous and heterogeneous communities.

1. The *agent* offers basic capabilities, such as autonomy and persistence, but not necessarily mobility and communication with users.

2. A homogeneous set of agents defines a *homogeneous community* which is the second level of the hierarchy.

3. In the third level of the hierarchy, the notion of *heterogeneous community* requires communication and interaction between a heterogeneous set of agents.

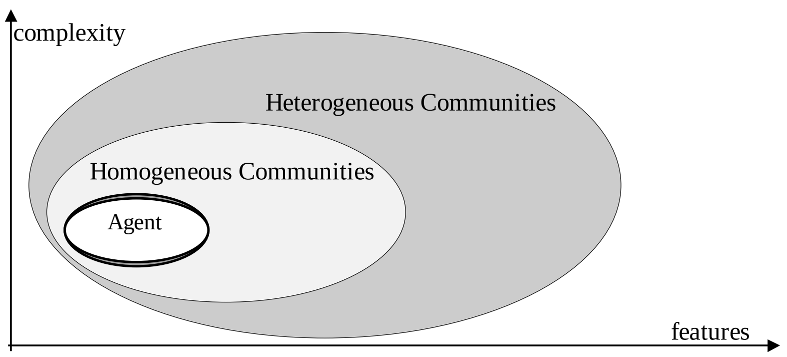


Figure 2: Relationships between an agent, and homogeneous and heterogeneous communities

3.4 More on Agents

With respect to mobility, there are two kinds of agents: mobile and stationary (or static) agents. In general, *stationary agents* are created in the context of a specific application at the user's initiative and become attached to that user for a long period of time. Since these agents do not move, they do not present special security problems to the system. However they should prevent (or be prevented from) attacks by foreign agents.

On the other hand, *mobile agents* are created by stationary agents, by other mobile agents, by other types of objects, or even by humans. They navigate between different places in order to execute some predefined tasks near to desired resources (such as other agents, a file system, or large databases). In this way, they may be useful when applied to mobile environments and applications, or in wireless networks with high-level latency and low-level communication rates [NC98, LO98].

3.5 End-Users

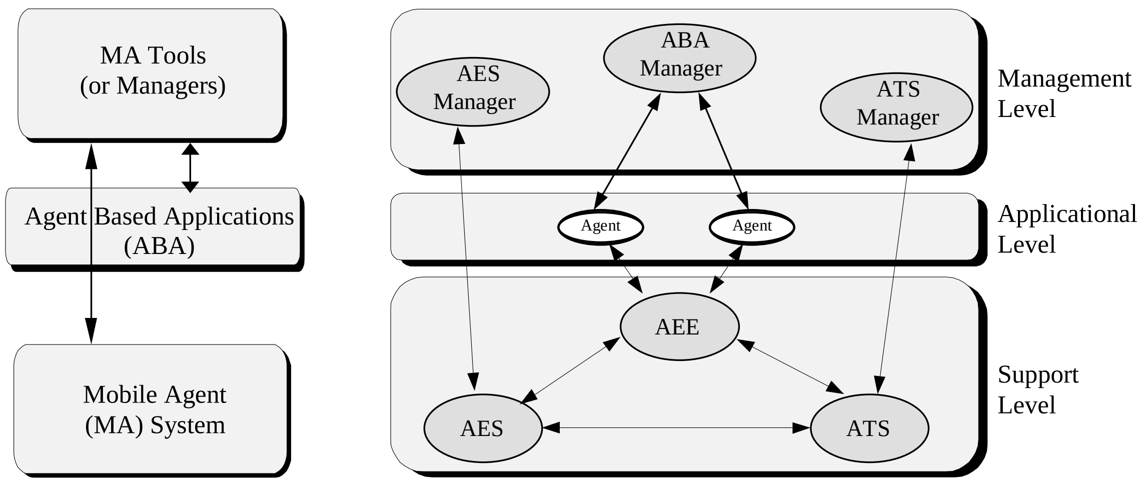
All *users* of an agent-based application have in general agents that execute on their behalf. Users can interact with their own agents, or can eventually interact with agents owned by other users if their respective security managers grant that behavior. Users interact with agents in several ways depending on their interface characteristics. For instance, they may interact through e-mail messages, HTML forms, AWT-base Java applets, or Active-X components.

Users have rights over their agents, namely to suspend, change their knowledge and goals, or even eliminate them. Nevertheless, these rights are restricted by the supported MA system as well as by the political rules of the respective application.

(Note that other kinds of agent applicability can be considered without a one-to-one mapping between an agent and a given user. Namely, in situations where a non-agent application can use agents to implement certain behaviors transparently.)

4. Generic and Global Architecture

In this section we introduce the main components of a generic and global architecture for supporting, developing and managing agent-based applications. This architecture is *generic* because its identifies several components that should exist from a conceptual point of view. It is *global* because it aggregates all the involved components needed to perform the required functions.



Compact View *Expanded View*

Figure 3: Generic and Global Architecture of the Reference Model

This architecture has three levels (see Figure 3): support, applicational, and management. The lower level of the architecture is represented by the agent support system (or MA system for short) which aggregates the following three components:

- The *Agent Execution System* (AES) provides support for at least the following operations: agent navigation and communication, persistence and security. The AES should have its own database in order to keep track of its internal resources, such as agents, places and thread pools. Additionally, the AES provides communication protocols to other AESs (with the same type, or with other AES types).
- The *Agent Type System* (ATS) keeps a dynamic set of agent types (or agent classes) in its own database, and provides an access interface in order for other entities (e.g., end-users, agents and specific tools) to reference or instantiate specific types/classes. Some ATSs can provide a higher-level interface based on the elementary agent types, with entities like “agent metaclasses”, or “agent profiles”.
- The *Agent Execution Environment* (AEE) provides an environment to execute agents. This component is typically the interpreter (or virtual machine) of the language used to develop the agents. There are also situations where the AES and ATS run on the top of some AEE (this situation was not represented in the figure for clarity).

Acronyms	**Expanded**	**Alternative Name**
MA System	Mobile Agent System	Agent System/Server
AES	Agent Execution System	
AEE	Agent Execution Environment	VM
ATS	Agent Type System	
VM	Virtual Machine	
ABA	Agent Based Application	Community
APL	Agent Programming Language	
ACL	Agent Communication Language	

So there is no confusion, we will use the term “MA system” generically to represent the three components: AES, ATS, and AEE. Although the majority of MA systems don’t handle this distinction, this separation is important whenever analysis, comparison, and evaluation are required.

At the intermediary level, we find the ABAs directly executed by some AEE. The ABAs are sets of interconnected agents.

Finally, at the upper level of the architecture, we find the following tools that provide users with the capabilities to interact and manage the MA system, as well as tools to develop and manage ABAs:

- *AES Manager* provides, through end-user interfaces, the capabilities to configure, manage, monitor and keep information of the respective AES.
- *ATS Manager* provides the capabilities to manage and keep information of the respective ATS.
- *ABA Manager* provides, through end-user interfaces, the capabilities to manage and monitor an arbitrary number of agents, namely the capabilities to create, activate, suspend, resume, dispatch to some place and send messages.

Similar to the lower level component of this architecture, the functionality of the above tools can be integrate in to just one tool. Obviously, there should exist some authentication mechanisms in order to enable different functions for different groups of end-users. The tools may be more generic or more specific. For example, you could consider one AES manager tool with the capabilities to transparently access and manage several AESs.

5. Technical and Functional Features for Mobile Agent Systems

MA systems have a significant set of architectural issues as well as technical and functional features in order to support agent-based applications. In this paper we focus on the following features: (1) execution, (2) management of agent types, (3) management of identifiers, (4) persistence, (5) navigation, (6) communication, (7) access to external resources and (8) security.

5.1 Agent Execution

The most basic MA system requirement is to provide an agent execution environment. This environment assigns each agent resources, such as disk, memory, CPU time, and communication channels. The AEE is responsible for the execution/interpretation of the agent's code. Therefore, it generally consists of a virtual machine or a specific interpreter, which conceptually tends to be an autonomous process of the corresponding agent execution system, as depicted in Figure 4.

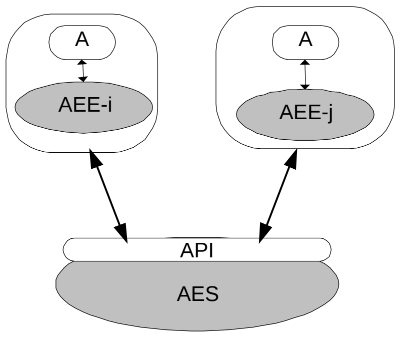


Figure 4: API for interaction between the AEE and the AES

There are two distinct approaches to MA system design:

- *MA system with proprietary Agent Programming Language* (APL). This is the approach used by most MA systems, in which there is only support for agents programmed in a specific programming language. The AEE is tightly integrated and optimized with the AES design and architecture (or vice-versa).
- *MA system independent of the APL*. The design of this kind of MA system (e.g., D'Agents and ffMain) permits the integration of different AEE in an independent and incremental fashion. This way, the MA system supports the execution of agents built using different programming languages, since the corresponding AEE exists. However, it should be noted that although agents may be programmed in different languages, they must provide a similar object model (e.g., libraries of specific classes).

Figure 5 illustrates variants to the conceptual model presented in Figure 4. This situation occurs when a virtual machine (e.g., Java) supports the AES. In this case, and when the AES and the agents execute using the same virtual machine, agents may be executed inside the AES context, as autonomous entities. The interface between the AES and its agents is comparatively more efficient and easier to implement, since it is based on direct invocation of methods known among the different entitiesexisting in the AEE/AES process (situation 1).

On the other hand, a generic scenario demands a physical independence between the AES and AEE/agents processes, so that the former may support agents, no matter which APL is adopted. Although the API is conceptually equivalent in both solutions, the implementation becomes more difficult and less efficient in situation 2.

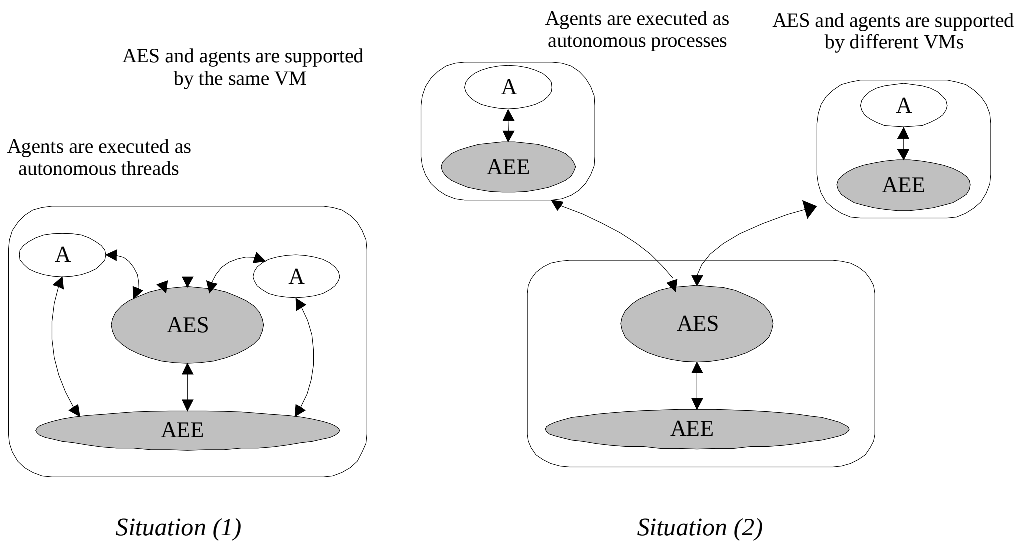

AES and agents are supported by the same VM
Agents are executed as autonomous threads
A
A
AES
AEE
Situation (1)
Agents are executed as autonomous processes
AES and agents are supported by different VMs
A
AEE
A
AEE
AES
AEE
Situation (2)

Figure 5: Relationships between AES and AEE

It should be noted that, although Java is currently the most popular language for developing MA system and agents, any other language could be used (preferably based on objects and with equivalent characteristics), as long as the equivalent APIs are provided. Typically, it is the AES's job to determine how a process executes when the agent is created, taking into account the information provided by its class. There are three possible situations:

- The virtual machine (VM or AEE) executing the agent class is compatible with, or equivalent to, the VM executing the AES, and the agent class is based on the available API. In this case, the agent executes as one or more autonomous *threads*, inside the computational context of the AEE, in conjunction with the AES (Figure 5 – situation (1)). This is the case of current Java based MA systems, such as Aglets, Mole, or AgentSpace.
- The language in which the AES is developed is either non-interpreted, or is not executed by any VM; or its VM is not equivalent to the one executing the agent class. In both cases, if the agent class' API is supported by the AES, then the agent is executed as a *process*, different from the AES (Figure 5 – situation (2)). This is the situation of systems such as D'Agents or Tacoma.
- A special case of the situation (2) is where the MA system as well as the different AEEs and agents run inside a single process. This is the case of the Ara [PS97] system.

5.2 Management of Agent Types

Since an agent is an instance of its respective type, it is necessary to understand how agents are instantiated.

In some approaches (e.g., ffMain, D'Agents) the notion of type/class of an agent is missing – that happens in the case of non object-oriented agent programming languages. If the notion exists, it is not usually independent from the applications that use it. In other approaches (where the agent programming languages used are inherently object-oriented), the classes are stored in some local/remote file system in the form of a file, without any particular organization or control. For example, in Java `Xpto.class` represents the `Xpto` class, and jar files, such as `ypto.jar`, can be used to represent the set of classes, images, and serialized objects, compacted according to the Java Archive format.

In next generation of MA system architectures, it will be important to have a system component exclusively responsible for the management of agent types/classes. This component, beyond the storage of a variable number of classes in its local database, must also provide mechanisms for accessing and searching for types/classes. Advanced services, such as distributed naming systems or proxy agents for agent types, can be built over the basic functionality of an agent type system.

Furthermore, specific information concerning agent types may also be described as a set of meta fields, such as author name, version, release date, supported ontology, or supported message types. This information will be called by agent meta classes or agent profiles, and managed in the ATS context.

5.3 Generation and Management of Identities

The interaction between agents and between agents and users raises the questions of how do agents identify and reference themselves and how do they identify the execution places in which they exist? We analyze these issues from different perspectives:

- *Type of identifier*: A resource can be identified in different ways: by address or by name. Generally, an *address* corresponds to the notion of an access identifier to the resource. It may be a direct or indirect reference to the object, and it may include the electronic address of the node in which the object lives. A *name* is an alphanumeric identifier, used to characterize the resource. To use name-based identification, there has to be a naming service that provides the necessary conversion to the corresponding address.
- *Management models*: There are two main models of address/name space management: linear or hierarchical. In *linear management* (e.g., Telescript and D'Agents), each resource is assigned a globally unique identification, independent from the place where the resource lives. There is a single service/server responsible for the assignment of unique addresses, and for the conversion from logical names to physical addresses. This approach is adequate for proprietary applications, with a small number of resources or with low-levels of distribution. In the *hierarchical management* model (e.g., Aglets and AgentSpace), of which DNS [Alb+92] is a good example, there is a hierarchy of domains and resources, each one managed by its own server. An address is composed by a sequence of components, each representing a sub-address in the domain hierarchy. Although this approach requires a more complex MA system, it is good for open applications, with many resources or with a high-level of distribution.
- *Support for heterogeneity*: While proprietary MA systems may easily adopt specific and optimized address management schemes, with additional functionality such as name services and access control mechanisms, this is not true for open systems. In these environments, the execution place and agent address management model must be hierarchical, based on the (also hierarchical) network node address model. If requested, the naming services should also be seen as DNS extensions.
- *Kinds of resources*: In general, the following resources must be identified distinctly: mobile agent systems, execution places, agents, and the types/classes of agents.

5.4 Persistence

Persistence can be defined as the capability of saving state information from one execution environment to another. A persistent agent, for example, outlives the operating system process in which it was created. But during its life cycle, an agent also accesses, manipulates and transports persistent data (or information) so the issue of persistence may be analyzed according to two perspectives:

- Agent persistence
- The way agents access (external) persistent resources (e.g., databases).

In this section, we concentrate on the first perspective. We discuss the second perspective in Section 5.8 since that section's focus is on the interaction between agents and external resources.

A persistence service for agents is necessary for different purposes. For example, it is required for process management, e.g., an agent inactive for a long period may be temporarily stored on disk to consume less memory. It is also required for fault recovery, e.g., to recover the state of an agent from a MA system, or even a node, that suffered a fault.

Having agreed on the need for persistence, there are several levels of persistence support:

- *External (or explicit) persistence*: Persistence is supported by a storage system outside the MA system, e.g., a file system or a database system (based on relational, object-oriented, or any other technology). The MA system may have to code/decode the agent (between the format in memory and the format of the storage system) whenever it is stored/retrieved. (This process is similar to agent migration between MA systems, consisting of the serialization (also called linearization) of the agent in a sequence of bytes in a predefined format.) The agent is serialized before being stored (or transmitted across the network), and the inverse operation (object deserialization) is performed when the MA system retrieves the agent from the store (or from the network). This functionality exists already in JDK 1.1 [Sun98], where simple mechanisms for object serialization are provided (however, it does not support thread-state capture without changing its virtual machine).

- *Internal (or implicit) persistence*: Internal persistence is defined as a mechanism automatically supported by the MA system so that an agent programmer does not explicitly use it or is even aware of it. For example, if the MA system is implemented on top of Pjama, then the system will automatically support internal persistence. However, the MA system itself can implement this functionality, maybe changing the Java virtual machine if necessary.

It is convenient to analyze the types of persistence that MA systems provide. The following approaches are the most common:

- *Only data*: The data manipulated by the agent (which may consist of complex data structures) is stored. The agent's data consists of a set of local, shared and global variables it manipulates. Unfortunately, not all variables are serializable. For example, an object serialized in JDK 1.1 does not include variables declared as *static* or *transient*. There are some problems regarding the transitive closure [MS97].
- *Only code*: The agent's code is stored, including references and code for inherited and/or referenced classes (i.e., the code's closure). The same problems with closures apply [MS97].
- *Only the image*: The execution state (i.e., information about the execution stack) is stored alongside the agent's data, but not the code.
- *The entire agent*: The agent is stored together in a consistent way, including data, execution image and specific code.

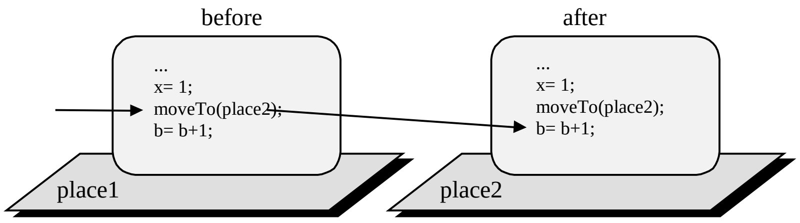
AgentSpace stores the agent's code and data. Storing the agent's state is far too difficult to implement and of little practical use. Furthermore, storing just the state without the code does not make much sense. So in our opinion there should be the following types of persistence: data, code, data and code, data, code and state.

5.5 Navigation

Mobile agents move explicitly between an arbitrary set of execution places. Nevertheless, each MA system, together with the respective model and programming language, supports navigation primitives that present distinct syntactic and semantics models. (In this paper we adopt the term "navigation" to mean the equivalent term "migration". The reader should be aware that they represent the same feature. However, we prefer to use navigation instead of the migration, because the latter is used more often in the context of operating systems processes, while navigation is used more often in the context of mobile agents. There is a subtle difference between both concepts: in the migration, the migrated process has no control of that action, while with mobile agent, it has control and autonomy to decide to move or not).

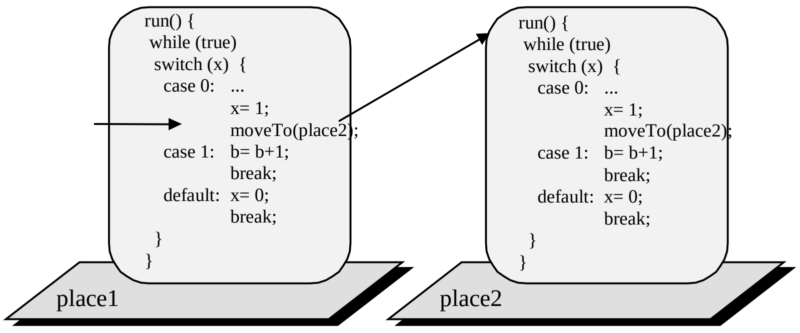
From the point of view of the execution state there are two approaches (see Figure 6):

- *Statefull (or strong navigation)*: In this class of MA system (Telescript, D'Agents) agents can contain navigation primitives at any point in the code, without requiring explicit state retrieval. When a navigation primitive is executed, the agent's current state (data and execution stack) and code are marshalled and transferred to the destination place. On arrival, the agent resumes its execution at the instruction following the navigation primitive. State retrieval is done by the AES component, and includes the following steps: (1) reception of the marshalled agent, (2) unmarshalling, and (3) re-creation of the agent's prior execution environment. In this model, although there is an increase complexity in the MA system, navigation semantics are more simple to use and transparent, thus making mobile agent programming easier. On the other hand, the fact that the MA system has access to the agent's execution stack raises some security and complexity issues (see Sections 5.8 and 5.9).
- *Stateless (or weak navigation)*: In this kind of MA system, an agent is transferred but its state of execution is not. This is the case in the majority of MA systems based on Java. In its current version, the Java virtual machine doesn't grant access to the running stack of an thread. For this reason, the MA system designer has to provide mechanisms to keep the state of a prior execution. A typical solution is to invoke a previously agreed callback (e.g., the agent's `run` method). This method is invoked after the navigation operation is performed. This solution (e.g., in Aglets) turns the agent's main method into a dispatcher for auxiliary routines, forcing a less elegant and more difficult to maintain programming style.



```
...
x= 1;
moveTo(place2);
b= b+1;
```

(1) Statefull navigation



```
run() {
 while (true)
  switch (x)  {
    case 0:   ...
              x= 1;
              moveTo(place2);
    case 1:   b= b+1;
              break;
    default:  x= 0;
              break;
  }
}
```

(2) Stateless navigation

Figure 6: Different semantics offered by the navigation operation

Figure 6 illustrates an example of the differences between two agents having a `moveTo` primitive with different semantics. The control of the execution in Figure 6-situation (1) follows the normal sequence (thus, it's statefull), while the agent in Figure 6-situation (2) is reinitialized (thus, it's stateless) after migration.

On the other hand, an agent's code consists not only of the code of its specific type, but also the code from all of its inherited and/or referenced types – the code closure. Therefore, from the code migration point of view, there are three different approaches:

- *The code doesn't migrate*: In this case, the agent's code has to be preloaded by the nodes the agent might visit, and only the execution image (i.e., data and stack) migrates. This approach is adequate for proprietary applications running on well-defined places, in which the types of supported agents are known in advance. Although this provides a pragmatic mechanism for agent migration, it lacks the flexibility that is required in the context of more dynamic, open applications on the global Internet.
- *The entire code closure migrates at once*: The agent's code is self-contained and self-sufficient. The main disadvantage is the fact that mobile agents with additional capacities (i.e., not supported by the MA system) may become relatively large; thus migration becomes a heavy and slowly operation (in terms of the required resources), and should be avoided whenever possible.
- *Code migrates incrementally*: Only the basic, specific agent code migrates. The remaining code (i.e., additional types, inherited or referenced) migrates and is loaded dynamically (on-demand) at runtime. This approach allows a more efficient and versatile agent migration, but introduces additional security control and agent autonomy problems.

5.6 Communication

With respect to communication and interaction between agents, it is important to discuss issues concerning the semantics.

- *Synchronically*: Communication is *synchronous* (blocking) when the parties involved synchronize themselves before transferring any data. This kind of communication is adequate in scenarios where data transfers requires urgent confirmations, or in interactive dialogues. Communication is *asynchronous* (non-blocking) otherwise, i.e., when the parties don't need to meet. *Deferred* (or *future* based) communication [Lis88, WFN90] is a hybrid model, in which the sender doesn't block automatically, and continues its activity until the desired results are available or until the sender decides to block itself.
- *Locality*: According to a spatial (geographical) analysis, interaction between agents may be local or remote. When interaction is *local*, agents meet at a common place and may use any known interprocess

communication mechanism, such as shared memory, files, environment variables or pipes. When interaction is *remote*, agents communicate from distinct nodes through message passing, RPC, or any other remote communication mechanism.

- *Intermediation*: Interaction between agents may be direct or indirect. In *direct* interaction, agents invoke each other's methods explicitly, whether or not restricted by additional security mechanisms. When interaction is *indirect*, agents don't communicate directly. Rather, they use an intermediation service that generally offers some kind of added value, such as programming language or MA system independence. Examples of this kind of service are object request brokers (ORB) or servers of shared information spaces, inspired by the Linda model [Gel85] (e.g., ffMain [LDD95], or PageSpace [CKTV96]).
- *Destination*: In general, when an agent interacts with other agents, it follows a *point-to-point* communication pattern. However, there are situations in which an agent interacts with a dynamic set of agents using *group communication*. If the choice of this set is restricted by certain parameters, then it is called *multicast*, otherwise it is called *broadcast* communication [Mul93, CDK94].

5.7 Communication during Navigation Operations

An important question in terms of communication and interaction between agents is: What happens when one agent, involved in some conversation/interaction with other agents (or objects in general), decides to migrate to another execution place?

This issue is illustrated in Figure 7, and is called the "open channels in navigation operations" problem. It was raised before in the context of operating system process migration [AF89, Wel90]. If agent A_1 is interacting with agent A_2, what happens to the communication channel when A_1 moves from $place_m$ to $place_k$?

The goal is that navigation operations don't interfere with agent communication. The term "channel" above stands for any communication means (e.g., sockets, or shared memory) between agents. This issue is particularly important in the direct interaction model.

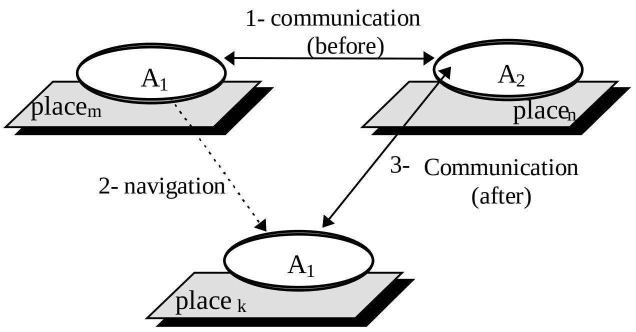


Figure 7: The Problem of the open channels in navigation operations

There are two approaches to solve this problem:

- *Channels are not kept opened*: The MA system simply doesn't keep the channels open. This means that when an agent migrates to another node, it looses the connections to its peers. In this situation, it is the user's (better yet, the programmer's) responsibility to deal with exceptions (e.g., by re-opening the channels), if needed or wanted. Before migration, the agent has to close all connections and keep a data structure with enough information to be able to recover the state of those connections later. However, it may not be possible, or extremely complex, to recover the state of communication channels.
- *Channels are kept opened*: The MA system is responsible for keeping the communication channels between agents open, providing a high-level of transparency (from the programmer's point of view). A possible approach to implementing this mechanism is based on the proxy model [Sha86, SDP92]. Entities don't interact directly with an agent, they do it through the agent's proxy. When the agent moves to another node, it leaves a proxy to receive the messages addressed to it; the proxy then forwards the messages to the agent, as soon as it is established in the destination node. This gives full transparency to the programmer. There are two ways to implement this mechanism, both of them also used in name management systems:
  - *Centralized proxy*: There is only one proxy object, generally kept in the node in which the agent was created. In this case, any navigation operation implies a notification of the proxy, both before and after performing the operation.

- *Chain of proxies*: When the agent migrates, it leaves a proxy behind that is responsible for receiving messages from the previous proxy, and for forwarding them to the next proxy, and to the agent itself.

5.8 Access to External Resources

Agents must be able to interact with external resources, such as database management systems, file systems, user interface devices (e.g., monitor, mouse, keyboard), and communication channels (e.g., supported at socket level). It is the responsibility of the MA system to provide agents with controlled access to the existing external resources. A MA system developer can consider the following conceptual approaches for this access:

- *Only static agents may access critical resources*. In this situation, mobile agents play a limited role, being able to interact with only other agents (mobile or static), and not directly with users. Static agents are responsible to explore the capacities of the various resources, and become mediators between the interactions between users and mobile agents, and between the latter and resource managers (e.g., database and file management systems). This approach restricts mobile agents' flexibility, but provides a simpler agent-based application development model, and simplifies the required security mechanisms.
- If the MA system provides the necessary APIs, mobile agents may – as long as authentication and access control mechanisms are preserved – access external resources. In this relatively liberal approach, mobile agents are extremely active and versatile computing components. However, they are also potentially dangerous – this is why they may never be used in open, heterogeneous environments. For this reason, it is fundamental to have an adequate and flexible integration with security mechanisms in future MA systems. We discuss the main security issues in the next session.

5.9 Security

Security is a critical issue for the process of mobile code execution, and in particular execution of mobile agents.

If the "simple" loading of mobile code (e.g., Java applets or Active-X controls) raises many security issues and challenges, agent security is even more serious in the presence of mobile agents, moving around an arbitrary and dynamic number of nodes, and communicating with an undetermined number of different computing entities.

The following security aspects must be considered by a MA system:

- *Authentication* and *access control* (prevention against attacks from malicious agents): When an agent migrates to a new node, its identity must be verified. The identities of its owner and/or the owner of the agent's original execution place may also be verified. If the authentication succeeds, the agent is executed by the MA system, but limited by the access/permission level granted. Permission management is performed at every execution place on a case-by-case basis subject to each organization's internal rules in the context of each application. The most usual processes of identify verification are based of digital signature mechanisms, cryptographic keys and certification authorities [Sch95, FB97].
- *Integrity*: While an agent (i.e., its encoded representation) is transmitted through the network, its contents may be tampered with, in such a way that its original goals are altered, or unwanted behavior added. This situation demands a "secure channel". This channel may be implemented in the following ways:
    - At the application level supported by the MA system, with mechanisms such as PGP [Zim94], or symmetric key cryptographic algorithms such as RSA [RSA78].
    - At the network level, with mechanisms such as SSL [FKK96] or TLS [DA98].
- *Privacy* and *Integrity* (prevention against attacks from malicious MA systems): Since mobile agent code is interpreted and the MA system has access to the agent's internal structures (code, data and stack), a malicious MA system can introduce a virus or steal resources (e.g., CPU time, electronic money, or credit card numbers). This issue is not, in general, addressed by agent systems, mainly because both applications and MA systems are proprietary and developed in well known and restricted contexts. However, in open and heterogeneous environments, the problem must be solved. One possible solution is the validation and certification of the MA system and/or the execution places, based on the digital signature mechanisms mentioned above. This way, an agent only goes to a remote place if it is certified by an independent and trusted certification authority that the agent recognizes.
- *Control of Resources consumption*: This aspect concerns the problem of how a MA system can control the consumption of resources from external agents. Even if an agent can't access critical resources of the

system (such as standard output, disk, or communication channels), denial of service attacks by the consumption of common resources such as CPU time or memory space should be prevented. By not considering this issue, a set of malicious mobile agents could decrease the overall performance of a system by simply consuming its CPU time.

Agent programming languages and interpreters may additionally provide secure execution at different levels. For example, the Java security code model is based on three complimentary levels: language level, code execution/interpretation level, and application and class library level.

6. Survey on Mobile Agent Systems

Having in mind the sections above, it is now possible to analyze and compare well-known MA systems according to the terminology, concepts and architecture proposed as part of the reference model.

We organize MA systems into the following categories, each represented by a MA system.

- **Telescript** [Whi94, Whi97, GM96a] is an example of a proprietary agent system because it is a closed system. It was developed from scratch and is one of the classical references for mobile agents systems.
- **Aglets Workbench** [IBM97, LO98] is an example of a Java-based MA system. Alternative systems include Mole [SBH96], MOA [MFC98], Odyssey [GM97], or Grasshopper [IKV98].
- **ffMAIN** [LDD95] is an example of an open mobile agent system, i.e., supporting standard Web technology, in particular the HTTP protocol. An alternative system is PageSpace [CKTV96].
- **D'Agents (**formerly called **Agent-Tcl)** [Gra95, Dar98] is an example of a system whose design was independent from any programming language at least in theory. An alternative system is Tacoma [JRS95] or Ara [PS97].

In this analysis we also include the AgentSpace system [SMD98, SD98] because it was designed mainly by the first author and was the genesis of this paper.

Of course there are many other MA systems, some developed in academic environments and other developed by the industry. Nevertheless, some of these systems have already disappeared (such as Telescript, Kafta, or Odyssey), others will disappear, while other should emerge in a near future. For a more complete and up to date list of MA systems the reader should consult the electronic catalogues noted in the Appendix, specifically the Mobile Agent List [MAL99], and the AgentBuilder’s list of agent tools [ACT99].

Furthermore, there are other systems that can not be defined as “true” MA systems even though they provide some code mobility features. Examples of these systems are the Voyager Universal ORB [obj99] and the Jumping Beans system [AdA99]. In Section 6.6 we describe some architectural aspects of these hybrid systems.

6.1 Telescript

Telescript [Whi94, GM96a, Whi97], developed by General Magic, was the first commercial product to provide a technology for distributed application development based on mobile agents.

The Telescript’s model was inspired in the electronic market paradigm, composed by two main types of participants: customers, and service providers (producers or brokers). Customers send agents to visit a set of places where they may find the services provided by the service providers.

The notions of mobile agents and execution places (simply called *places*) are very clear. In Telescript, a place is an operating system process that provides not only a context for agents but also provides services. Therefore, a Telescript place can be characterized, based on our reference model, as a set of a place and a stationary agent.

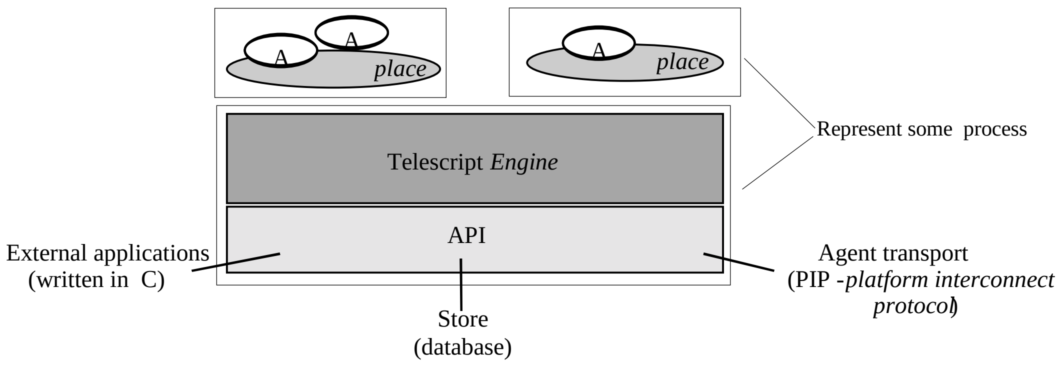

A
A
place
A
place
Represent some process
Telescript *Engine*
API
External applications (written in C)
Agent transport (PIP -*platform interconnect protocol*)
Store (database)

Figure 8: Conceptual schema of the Telescript framework

Telescript is mainly composed of the following components:

- The programming language (also called Telescript), heavily object-based and similar to Smalltalk, in which "everything" is an object, dynamic, persistent, portable and secure.
- The MA system, called the "engine". This component executes the Telescript language, i.e., it supports and executes the resources from the object model, such as agents and places. The MA system also manages those resources via three APIs: persistence management, agent transport, and access to external resources (see Figure 8).
- Protocols to represent the agents (i.e., object serialization protocol) and their exchange between Telescript engines.

Telescript was designed for the development of homogeneous ABAs, in proprietary environments, such as services in telecommunications networks managed by their respective operators. AT&T's PersonalLink – for PDAs – was one of the few examples this technology.

Later, Telescript adopted the open environment of the Web with a system called Tabriz [GM96b], but without much commercial success. Tabriz was composed of a set of services, which integrated Telescript and Web technologies, based on well-known server-based Web models [SMD97]. In 1997, General Magic announced that it has abandoned the Telescript/Tabriz products. At the same time, General Magic announced its new Java-based MA system, called Odyssey [GM97], which adopted some of Telescript's concepts and techniques. Nevertheless, today Odyssey is also a defunct project. It should be clearly stated that Telescript, Trabiz, as well as Odyssey were abandoned not only because of their technical problems, but mainly because of business shifts at General Magic.

As can be seen in Table 2, Telescript was an advanced MA system in several areas, such as security, mobility, persistence and communication. However, it was not flexible and dynamic enough, since it was designed for proprietary applications in closed environments. For example, name and address management was not based on Internet standards; direct interaction among agents implied they had to know each other at development time. As such, it was not possible to introduce new agents, with new interfaces and functionality, at runtime. Even worse, Telescript was a huge and unstable system with poor performance.

6.2 Aglets Workbench

Aglets Workbench (AWB) or simply Aglets [IBM97, LO98] is a MA system conceived at the IBM Tokyo Research Laboratory and is supported by the Java technology. It contains the following major components:

- JAAPI (*Java Aglet API*): Consists of a set of classes and interfaces that allows the construction of agents and applications based on those agents.
- Aglet Server (*agletsd*): A Java application that is divided into the following components: (1) server of the proprietary ATP protocol (Agent Transport Protocol) on which navigation and agent communication operations are supported; (2) the Aglets execution context; and optionally (3) a GUI aglets monitor. This tool known as Tahiti, provides a graphical interface (based on the Java AWT package) for management and monitoring aglets.
- Fiji: Allows the creation of applets with support (i.e., execution contexts) for aglet existence. This component allows users to dispatch aglets from any Web browser (provided these browsers run Java applets).

The Aglets object model includes a set of classes and interfaces contained in the JAAPI library. An aglet is an (potentially mobile) object that exists and is executed, at each moment, in some determined context (`AgletContext`). Each aglet has one shadow object representing it, called `AgletProxy`. This proxy functions as a shield and mediates all accesses to its respective aglet. The proxy prevents direct access to the aglet's methods and provides location transparency.

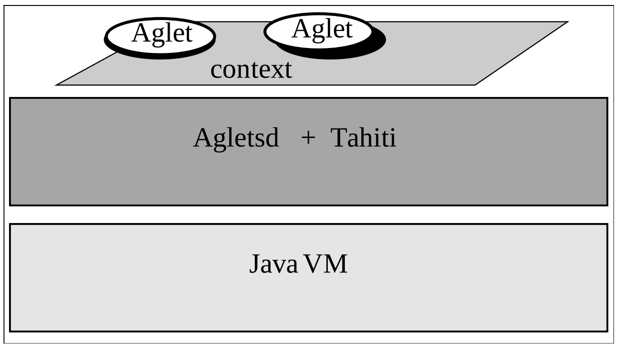


Figure 9: Conceptual schema of the Aglets framework

From a technological perspective, Aglets have the following main features:

- *Identities*: The management of Aglets server (or context) identities is based directly on the URL (uniform resource locator) mechanism. On the other hand, aglets themselves are identified through an `AgletIdentifier` instance, which is guaranteed to be unique and global for each aglet.
- *Persistence*: Although not explicit from the available documentation, external persistence for the aglets's data is provided. On the other hand, the code is kept on disk in the same way that any other Java class is stored.
- *Navigation*: In part due to the fact that the current Java virtual machine specification does not provide access to the stack of the current running thread, Aglets don't provide statefull agent navigation. This means the programmer has to control from what place the agent is coming using global state variables.
- *Communication*: The proxy model minimizes the problem regarding communication between mobile agents. As briefly explained above, agents communicate between one another using their respective proxies. Additionally, several communication semantics are provided, such as synchronous, asynchronous, and future-based semantics.
- *Access to external resources*: Since aglets are Java objects, there is no interaction restrictions between them and other external resources provided they exist and aglets know the respective APIs (e.g., AWT, JDBC, RMI). Additionally, the security manager, defined in the global server level, allows every type of interaction for agents created locally (see below).
- *Security*: A security manager (an instance of the `AgletSecurityManager` class) is defined to recognize two types of aglets: *trusted* and the *untrusted*. An aglet is trusted when it has been created locally and its codebase is local. If these assumptions are not verified, it is not trusted. The server manager can configure, based on a predefined set of categories (e.g., file system, network, windows system), the different operations not allowed by untrusted aglets. On the other hand, trusted aglets have access to the majority of the operations provided by the server.

6.3 ffMAIN

The ffMAIN [LDD95] project was developed in the Göethe University, Germany. The main importance of ffMAIN its mobile agent paradigm for open environments and its exclusive use of Web technologies.

The general architecture of this system contains three components, as shown in Figure 10: agent server, execution environment, and agent. The agent server consists of a HTTP server modified to handle a new set of methods (in addition to the GET, PUT, and POST) and to keep an information space to support communication between agents.

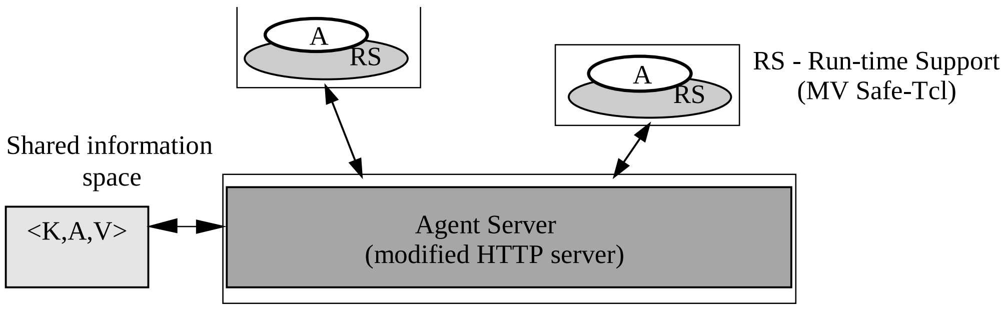


Figure 10: Conceptual schema of the ffMAIN framework

The execution environment consists of a virtual machine that supports execution and interpretation of the agent code. In the concrete case, a Tcl interpreter was adopted [LDD95]. However, according to the authors, there are no restrictions that prevent other languages from being supported. The agent server creates an autonomous process for each agent and its respective execution environment. Independence between the MA system and the agent is obtained in this way.

The process of transporting agents is based on the HTTP protocol. When the user wants to create an agent, a HTTP message is sent. This message contains the information `POST` *`server/`*`create` (where `server` it is the URL of the extended HTTP server) in the header and the agent data, code and attributes (encapsulated in format MIME) in the body. When the server receives this message, the message is validated (on the basis of the indicated attributes), and in case that it is valid and contains a recognized agent, the server dispatches that agent's execution to the appropriate environment. Finally, the server returns a reply message to the user indicating the result of the operation.

The agent management mechanism (suspension, parameters update, removal) is done by the user or by another agent through specific messages sent to the server (POST, GET, and DGET methods). For example: `POST server/create`, `POST server/moves`, `POST server/visit/id`, `POST server/info?key`, `GET server/info?key`, `DGET server/info?key.`

Agent communication is done indirectly through elementary operations of writing (POST), reading (GET), and destructive reading (DGET) in a public information space as proposed by the Linda model [Gel85, Fre96]. This space is kept and managed in each node by the respective agent server and consists ofa set of tuples in the form <*K*,*A*,*V*>, where *K* is the key (identity); *A* is an access control list that specify all the operations and entities allowed to make operations on the tuple; and, *V* consists of the information itself, whose syntax and semantics are defined in the context of each application.

6.4 D'Agents

D'Agents (formerly called Agent-Tcl) [Gra95, Dar98] is a mobile agents infrastructure (in their own original definition: "transportable agents") that was developed at Dartmouth College, in the USA. Although it was developed based on a Tcl interpreter, D'Agents was designed to be independent of virtual machines and their respective languages. Currently, D'Agents supports, in addition to Tcl, Scheme, Java, and C/C++ for stationary agents.

D'Agents does not consider the notion of an execution place as a structured form of meeting and execution of agents – the execution place corresponds in D'Agents to the identification of an agent server.

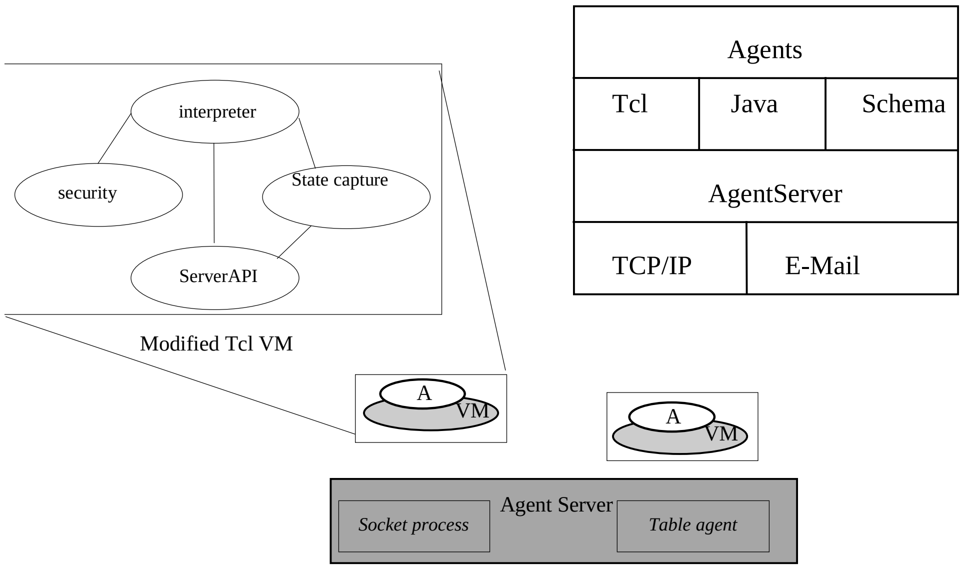


Figure 11: Conceptual schema of the D'Agents framework (applied to the Tcl virtual machine)

The Tcl interpreter consists of one extended Tcl virtual machine with the following four components (as shown in Figure 11): (1) a module to capture the agent state in order to provide transparent checkpoint/restart in navigation operations; (2) a security module, in order to inhibit agents from executing improper operations; (3) an interface (API) for the agent server to implement, containing, for example, communication and navigation primitives; and (4) the Tcl interpreter.

The agent server consists of two co-operative processes: (1) a *socket process* that is responsible for the creation, management and transport of agents, supporting, for example, the primitive `agent_fork`, `agent_jump` and `agent_submit`; and (2) an asynchronous process *table agent* that is responsible for the management of identities and the communication between agents through the primitives `agent_send` and `agent_receive`.

Each agent is an autonomous process, created by the agent server, that beyond its specific code, has the code of its respective interpreter. Consequently D'Agents provides (such as ffMAIN), a true independence between the AES and the language in which an agent is developed.

6.5 AgentSpace

AgentSpace [SMD98] is a Java mobile agent framework developed at the Technical University of Lisbon (IST/INESC), in Portugal. The generic and global architecture presented in Section 3 motivated the development of AgentSpace. Therefore, it is natural that AgentSpace fits the proposed reference model quite well.

AgentSpace's main goals are the support the development and management of dynamic and distributed agent-based applications. These goals are provided by three separated but well-integrated components, as depicted in Figure 12.

Both server and client components run on top of Voyager [Obj99] and the Java Virtual Machine (VM), and they can execute in the same or in different machines. Agents always run in an AgentSpace server's context. On the other hand, they interact with their end-user through applets (or frames) running in a Web browser's context or generic applications running directly on the Java VM.

The *AgentSpace server* (*AS-Server*) is a Java multithreaded process in which agents execute. The AS-Server provides several services: (1) agent and place creation, (2) agent execution, (3) access control, (4) agent persistency, (5) agent mobility, (6) generation of unique identities (UID), (7) support for agent communication, and optionally (8) a simple command-line interface to manage/monitor the server and its agents.

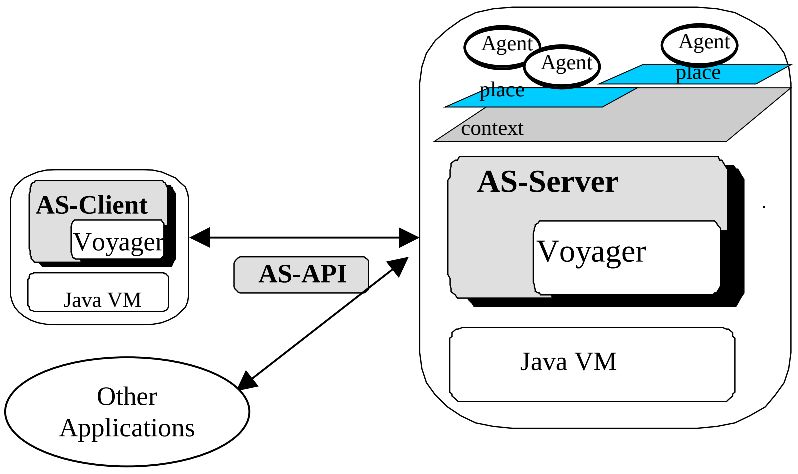


Figure 12: The AgentSpace Architecture

The *AgentSpace client* (*AS-Client*) supports – depending on the corresponding user access level – the management and monitoring of agents and related resources. The AS-Client is a Java applet/application (stored on an AS-Server's machine). It provides an adequate integration with the Web and offers Internet users the possibility of managing their agents remotely. Furthermore, the AS-Client is able to access several AS-Servers at the same time, providing a convenient trade-off between integration and independence between these two components.

The *AgentSpace application programming interface* (*AS-API*) is a package of Java interfaces and classes that defines the rules to build agents and to use other resources kept in the AgentSpace Server. In particular, the AS-API includes supports for the following: (1) building agent classes and their instances (agents), created and stored for later use in the AS-Server's database, and (2) developing client applets/frames in order to provide an interface to agents.

AgentSpace, like Aglets, provides an environment to develop Java-based applications. Although there are similarities between both systems, their respective object models and APIs are different.

The following differences between AgentSpace and Aglets, relating to programming elegance, security mechanisms, end-user integration, or location transparency, provide advantages for the programmer [SMD98, SD98]:

- *Keeping open channels*. Aglets does not provide this feature, while AgentSpace supports it transparently.
- *Agent References*. In AgentSpace it is possible to obtain a reference to an agent (`AgentView`) from its respective identifier (an `AgentID` instance). In Aglets, to obtain the reference (`AgletProxy`) it is necessary to have its respective identifier (an `AgletID` instance) and the URL corresponding to the place (`AgletContext`) where the agent is currently running. Additionally, the agent reference (`AgentView`) in AgentSpace is better than the equivalent in Aglets (`AgletProxy`) because it hides the agent's location and provides a flexible access control politic which is (an object) dynamically attached to the agent.
- *Specify callback methods after navigation operation*. Both systems provide a stateless (weak) agent navigation. However, in Aglets the system always invokes the same callback (`onArrival`) after a navigation operation, while in AgentSpace the programmer has the opportunity to specify, as a parameter of the `moveTo` method, the callback that should be invoked after the navigation operation. This subtle difference provides a simpler and more elegant solution and makes it easier to maintain the corresponding code.
- *Sending messages with objects of arbitrary types*. AgentSpace provides the capability to send messages with objects of arbitrary types, in particular types/classes that do not exist in the machine of the message receptor (in this case, those classes are loaded remotely from the message sender's machine). On the other hand, Aglets does not support this feature. It requires that all the types involved in a message must be installed in the corresponding machines before the message is sent.

Furthermore, AgentSpace presents a more complete object model by providing the following: (1) a better resource organization, through the notion of execution places, (2) the integrated management of users, groups of users, and permissions, (3) flexible security mechanisms that allow the dynamic attaching of security managers to agents, as well as to places, (4) the transparent association between a security policy, a user and an agent at it's the agent's creation time, (5) a suitable integration of agents with Internet/Web based applications.

On the other hand, Aglets has the following advantages over AgentSpace: (1) an elaborated communication and management message-based mechanism, (2) a flexible model to handling events, and (3) a significant collection of specialized agents, and related helper classes (e.g., KQML brokers; notification of events; meeting point). This last point is a consequence of its inherent popularity and its association with IBM.

AgentSpace and Voyager (see below for a brief description of the Voyager Universal ORB) are closely related. Naturally, some of AgentSpace's strengths are a consequence of the adoption of Voyager, which is a better middleware alternative than, for example, RMI. However, AgentSpace has its own virtues, specifically at the object model definition level, where there is a tight association between agents, users, and security managers at agent creation time.

6.5.1 Related Work

6.5.2 Voyager Universal ORB

Voyager is commercial product from ObjectSpace Inc. that supports the development of Java based distributed applications. Voyager is a 100% Java set of products. Originally (the first version), Voyager was just a Java ORB much better than the equivalent RMI from Sun. However, nowadays one should view the Voyager Universal ORB (version 3.1) as a product line with the following items [Obj99]:

- *Voyager ORB*. A high-performance, full-featured object request broker that simultaneously supports CORBA and RMI. Its innovative dynamic proxy generation takes ease of use to the next level by removing the need for stub generators. Voyager ORB includes a universal naming service, universal directory, activation framework, publish-subscribe, and mobile agent technology.
- *Voyager ORB Professional*. Builds on the Voyager ORB foundation with a configuration framework, COM support, JNDI integration, persistent replicated directory, CORBA naming service, Dynamic XML, connection management and support for ultra-light (15K) clients.
- *Voyager Management Console*. Allows the creation and maintenance of configuration profiles for Voyager servers. In addition, it provides the ability to monitor Voyager services, such as security and transactions, in real-time. The management facility is fully extensible, providing a framework for graphically managing user-defined objects from the console. (Requires Voyager ORB Professional)
- *Voyager Security*. Includes a flexible security framework, lightweight security implementation, support for secure network communications via SSL adapters, and firewall tunneling using HTTP or the industry-standard SOCKS protocol. (Requires Voyager ORB Professional)
- *Voyager Transactions*. Delivers full OTS-compliant distributed transactions support, including two-phase commit, a one-phase commit JDBC adapter, flat and nested transactions. (Requires Voyager ORB Professional)
- *Voyager Application Server*. Offers a true Enterprise Java Beans (EJB) development environment that decouples application logic from systems programming logic. Built on the strength of Voyager ORB Professional, Voyager Application Server goes above and beyond the Enterprise Java APIs to support interaction of a wide range of technologies. Voyager enables organizations to leverage their investments in technology by serving as a bridge between legacy APIs and the new Java Enterprise APIs. And because Voyager Application Server embraces the new Java Enterprise APIs, organizations are able to evolve with the industry.

It is important to note that Voyager is not a really MA system, as far as we discuss in this paper, having a broader focus and applicability. However, Voyager has the notion of an agent (as a autonomous and mobile object), as well as the notion of a place (an object or an application to where the agent can moves on). Voyager .

6.5.3 Jumping Beans

Jumping Beans is a commercial product from Ad Astra Engineering Inc. [AdA99] that allows Java developers to build mobile applications that can jump from computer to computer during execution.

Jumping Beans provides a different conceptual and architectural model, as shown in Figure 13, than the ones analyzed in this paper.

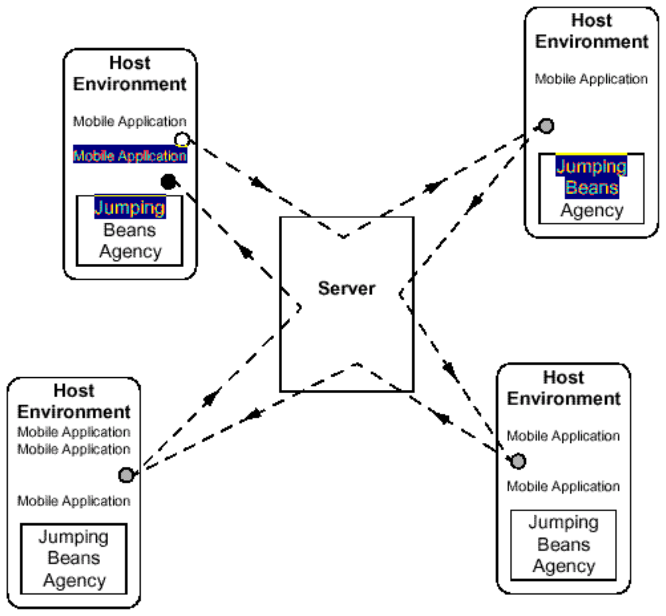


Figure 13: The Jumping Beans Architecture

Jumping Beans uses the terms "mobile application" and "Jumping Beans Agency" which correspond (more or less) to the terms "agent" and "context" used in our reference model.

Furthermore, Jumping Beans includes some innovative features at the technical level. While the majority of the MA systems are designed on top of a peer-to-peer architecture, Jumping Beans presents a client-server architecture. Clients are the facilitating software installed on each machine where mobile applications can run, and are called "agencies". The server is the software component responsible for the tracking, management and security control. Mobile applications never run on the server context. A set of agencies with a server constitutes a Jumping Beans *domain*. A domain has a central server, which authenticates the agencies joining the domain.

Other additional features of Jumping Beans are summarized in its White Paper [AdA99]:

- Jumping Beans' architecture, including its addressing scheme and Client/Server model, is fully scalable, whereas others have difficulties in scaling.
- Jumping Beans is easy to integrate into existing environments because it: (1) does not require migration to a proprietary ORB, (2) instantiates the arriving application into an ordinary Java VM, and does not require any proprietary execution environment, (3) requires no Java inheritance, and only one simple interface implementation.
- Jumping Beans mobile applications are easily managed from the central server. Other agent technologies make no provision for agent management.
- Jumping Beans has built-in security that is comprehensive and easy to manage. Some other agent technologies provide little or no security. Others offer security features that are difficult to manage and require a significant development effort on the part of the end user to implement.
- Jumping Beans is fault tolerant. In the event of an unexpected hardware failure, Jumping Beans will recover to the last saved state of all components. Also, the transfer of Jumping Beans mobile applications from one host to another is fully transactional, so that a mobile application will not be duplicated or lost in the event of an unexpected network failure.
- Some other agent technologies require an awkward post-processor. Jumping Beans mobile applications use natural Java classes and do not require any post processing.

- Jumping Beans easily addresses real world problems with many contingencies and complicated interrelationships, such as failing networks, security, disconnected operation, and management,. Other agent technologies have focused on solving ideal, rather than real world, problems. Therefore they are not designed to be easily implemented in a real enterprise-computing environment.

In spite of Jumping Beans claims, it seems clear that Jumping Beans' architecture has a central point of failure and a potential performance bottleneck at its central server. However, one approach to solve these problems is to create many small domains, each with its own server. Nevertheless, in the current version of Jumping Beans agents cannot migrate between domains. Jumping Beans claims it will support that feature in the future.

6.5.4 Concordia

Concordia is a Java-based, middleware infrastructure for developing and managing mobile agent applications. It was developed at Mitsubishi Electric ITA [WPW+97, ME99] and consists of a set of Java class libraries for server execution, application development and agent activation. Each node in the system, referred to as a Concordia Server, contains the following components running on top of a Java VM:

- Security Manager: This component is a Java object owned by the Java VM that manages resource protection. Agents are assigned identities that allow them to access server resources. Different from Aglets, resource protection is based on the user of the agent rather than the developer.
- Service Bridge: This component enables developers to add services to a Concordia Server to support agents when they navigate to the server.
- Directory Manager: This component enables agents to locate servers they wish to interact with on a host.
- Administrator: This component runs on a separate Java VM and is responsible for starting and stopping other components in the server. In addition, it manages changes in security profiles of both agents and servers, it monitors the progress of agents, and it collects agent and system statistics.
- Persistent Store Manager: This component is responsible for the storage of an agent's internal state and the recovery of an agent after a server failure or system crash.
- Event Manager: This component accepts event registrations from objects and agents, listens for and receives events, and notifies interested objects and agents (local and remote) of the events it receives – an application of the observer pattern.
- Queue Manager: This component manages inbound and outbound queues for the fast, reliable transport of agents across the network between local and remote Agent Servers. Agents are stored on the local queue while a remote server is under repair to provide a store and forward mechanism and are stored on local disk to implement the well-known handshaking protocol of Two Phase Commit, thus increasing reliability.
- Agent Manager: This component is responsible for propagating an agent to an Agent Manager at another Concordia server and the invocation of an agent's predetermined method (located in its Itinerary and configurable on a location basis) once it arrives. It also provides the execution context for an agent.
- Agent: This component is an object containing a combination of the agent's code and data. An agent's travels are described in its Itinerary, a collection of destinations defined as location-method pairs, maintained outside of the agent.

Concordia has several features worth noting. First is its itinerary scheme for dynamic invocation of an arbitrary agent method once it navigates to a server, which is similar to AgentSpace's callback approach. Second, in addition to managing asynchronous event communication between agents, is Concordia collaborative inter-agent communication. In this situation, CollaboratorAgent and AgentGroup classes are provided for agents to synchronize meetings, and analyze and share results. Finally, there is Concordia attention of failure support. In addition to making agents persistent, Concordia considers other situations such as manager failures, for example when the EventManager restarts, and situations when agents fail to arrive at their meetings. In these situations, the use of proxies has increased the reliability of the system by shielding agents and objects from the effects of server and system failures. To be an industrial success, future system must consider these and other quality of service issues.

7. Discussion

Tables 1 and 2 show a comparative analysis of the main aspects of the MA systems discussed in the previous sections. Table 1 compares the terms and concepts. Table 2 compares the main technical features.

7.1 Concepts and Terminology

Although the MA systems analyzed have many common or similar aspects, they also have important differences.

In Table 1 we identify the common aspects that all MA systems support generically:

- All require some kind of server (which we call in this paper “MA system”) that supports the main functionalities.
- All provide agents as autonomous and active components (or active objects) that can be autonomous processes, threads, or groups of threads, which are independent of the implementation level.
- All provide features to support the same kind of applications: *distributed applications based on the agent paradigm*, which are independent of their respective object model complexity, richness, or ease of use.

Table 1: Comparative analysis of MA systems’s concepts and terminology

Reference Model	Telescript	Aglets	ffMAIN	D'Agents	AgentSpace
Terminology					
Agent	mobile agent	aglet	agent	agent	agent
Place	place(1)	N	N	N	place
Context	place(1)	context	server(2)	server(2)	context
Support Level					
AES	Engine	Agletsd+Tahiti	HTTP server(3)	D’Agents server	AS-Server
AEE	Engine	Java VM	RS (Tcl VM)	Tcl, Java, Scheme VM	Java VM
Management Level					
AES Manager	N	Tahiti	HTML pages	Y	AS-Client
ATS Manager	N	N	N	N	AS-Client
Agent Languages					
Communication (ACL)	Telescript	Java + JAAPI	HTTP	Tcl/Java/scheme + Extensions	Java + AS-API
Programming (APL)	Telescript	Java	any	any	Java

Y/N - Yes / No.

(1) The notion of place in Telescript is slightly different of the proposal in the reference model. It corresponds to a process identified in the context in which the agents are executed.

(2) It does not have a proper class of context like those found in Aglets or AgentSpace.

(3) Modified HTTP server to support new methods (e.g., DGET) as well as to keep a shared information space.

The main differences between the MA systems analyzed are based on the following issues.

Terminology

Different names are used by different MA systems to refer to the same concept. For example, agents are called “mobile agent”, “transportable agent”, and “aglet”; agent execution systems are called “engine”, “agletsd”, “extended HTTP server”, and “AS-Server”. Moreover, the same name is often used with different meanings. For example, the Telescript’s “place” is better viewed as a set of a “stationary agents” than a place according to our reference model.

Support Level Issues

There are essentially three conceptual components in all MA systems. In general, the set of these components is called “mobile agent system”. The first component, agent execution system, is common to all the systems discussed. However, the other two components present significant differences:

- *Agent Type System* (ATS). The majority of current MA systems don’t provide any kind of support to create or to manage agent types based on, for example, meta or profile objects. AgentSpace provides this kind of support based on proprietary meta classes. Grasshopper, as an OMG’s MASIF compliant system, also provides this kind of functionality through the adoption of profile classes.
- *Agent Execution Environment* (AEE). All MA systems analyzed provide an environment where agents are executed. These environments are virtual machines of the supported programming languages, such as Telescript (for Telescript), Java (for Aglets and AgentSpace), and Tcl (for ffMain and D’Agents). However there is a fundamental difference between them as noted in Section 5.1: in MA systems like ffMain or D’Agents, agents run as autonomous external processes separated from their respective AESs. Systems like these are more modular and opened regarding multiple programming language support. On the other hand, in systems like Telescript, Aglets and AgentSpace, agents and the AES are executed in the same process because agents are implemented as threads or groups of threads. This situation implies a tighter coupling between AESs and agents/AEEs and consequently better performance.

Management Level Issues

As introduced in Section 3, the management of our proposed generic architecture involves a set of tools – also called “managers”. The main idea is to provide specialized tools associated with each component defined at the support level. Different users could access and manage different resources such as agents, agent classes, meta agent classes, places, or groups of users. These tools should be sufficiently generic to be used independently of any specific application domain.

We didn’t find any relevant information concerning tools for Telescript, due to it being marketed as part of the Tabriz web-server package and then withdrawn from the market, and due to the fact that the known tools are oriented to specific application domains.

D’Agents has extensive debugging and tracking tools. It also has managers to handle internal resources, such as consumables, files, libraries, programs and the network.

Regarding ffMain, the tools provided relatively simple HTML pages with direct access to the corresponding AES (extended HTTP server). Despite some useful functionality, this interface is too simple and restrictive from the end-user and programmer points of view.

With respect to Aglets and AgentSpace, both systems offer GUI tools based on Java libraries, which are called Tahiti and AS-Client respectively. Nevertheless, there are some differences between them. Tahiti runs as an agletsd’s (Aglets server) internal thread, so there is a one-to-one mapping between Tahiti and the server. On the other hand, the AS-Client runs as a Java applet or a stand-alone Java application. This implies that AS-Clients and AS-Servers can run independently on the same or on different machines because they interact through a local/remote communication protocol. Furthermore, the AS-Client allows users to connect to one or more AS-Servers anywhere on the Internet. Lastly, the AS-Client is the only tool that offers agent classes and meta classes management (i.e., it incorporates the ATS manager component).

Applicational Level Issues

All MA systems analyzed offer support for homogeneous agent-based applications. However, none of them consider the support for heterogeneous ABAs. Nevertheless, this aspect may be tackled through the adoption of some high-level communication mechanism (e.g., based on the KQML [LF97]). Another way to handle this issue, at the MA system architectural level, is through the adoption of some interoperation standard, such as the OMG’s MASIF [Mil98]. Grasshopper was the first MA system that implemented the standard.

Agent Languages

In this group we analyzed the relationships between MA systems and languages used to build agents. We identified two distinct types of languages:

- *Agent Programming Languages* (APL). These are the languages used to program agents. The majority of current MA systems use Java as the language to develop agents. This is the case for the Aglets and the AgentSpace systems but also for other systems such as Mole [SBH96], MOA [MFC98], Odyssey [GM97], JAE [PL97], CyberAgents [FTP96], Grasshopper [IKV98], Concordia [WPW+97], and even, more recently, D’Agents. However, approaches like ffMain and D’Agents that are open regarding agent programming languages can also (at least in theory) support Java. Nevertheless, in all cases there is a common object model with a proprietary API.

- *Agent Communication Languages* (ACL). The MA systems we analyzed have a common rationale to the ACLs: agents communicate using proprietary primitives provided by their respective MA system's object models. Because the focus of the MA systems were not on supporting heterogeneous ABAs, agents communicate among themselves using their own programming languages and those low-level primitives. For example, Aglets offers programmers a KQML Java-based package in order for agents to exchange KQML messages. We expect to see in the near future the majority of MA systems offering some kind of APIs based on KQML and/or FIPA's ACL.

7.2 Technical Features

Table 2 summarizes the main technical features of the MA systems analyzed. We have focused on the following aspects: identities, persistence, navigation, communication, access to external resources, security, and web integration.

Table 2: Comparative analysis of the main MA systems technical features

Reference Model	Telescript	Aglets	ffMAIN	D'Agents	AgentSpace
Identities					
Management type	linear	hierarchical	hierarchical	linear	hierarchical
Schema	proprietary	open	open	proprietary	open
Persistence	external	external	external	external	external
Navigation					
State	with state	without state	without state	with state	without state (5)
Code	not known	all	all	all	all
Communication					
Semantics	synchronous	sync, async, future	async	async	Sync, async,future
Locality	local, remote	local, remote	local, remote	local, remote	local, remote
Broker	direct	direct, indirect	indirect	indirect	direct, indirect
Open channel support	N	N	N	Y	Y
Access to external resources	via places	via Java API	via API supported by the RS	via API	via Java API
Security					
Authenticity and access	Y	Y (1)	Y(2)	Y	Y(3)
Integrity	Y	N	N	Y	N
Privacy	Y	N	N	Y	N
Resources consumption	Y	N	N	Y	N
Web integration	Y(4)	Y (Java)	Y (HTTP)	N	Y (Java)

Y/N - Yes / No.

(1) *Aglets only provides two levels of security: "trusted" and "untrusted" agents.*

(2) *The model of access control is defined at the Agent Server and and Run-time Support level.*

(3) *The model of access control is defined at different levels, namely at the context, place, and agent level.*

(4) *Later, through the Tabriz product line, the integration of Telescript with the Web was possible although in a restricted, non-natural way.*

(5) *AgentSpace doesn't support navigation keeping the agent state, but provides the possibility to the agent explicitly indicate the method that should be invoked after the navigation operation.*

Identities

As noted in Section 5.3, there are different tradeoffs and technical decisions regarding the definition of agent identities when designing a MA system. Telescript and D'Agents adopt agent identities in linear name spaces based on property schemas. All the other systems adopt an open schema based on uniform resource identifiers (i.e., internal URI/URL).

Persistence

All systems use an external repository to keep the required information. In Java systems (Aglets and AgentSpace) the agent's code is kept in the file system (in some directory referenced by the MA system's classpath). On the other hand, the agent's data (i.e., the serialized version of the agent) is kept in some proprietary binary file together with all the other resources manageable by these systems. Finally, in the majority of systems the agent's execution image is not kept because the Java virtual machine doesn't grant access to the thread's stack information. However, other systems (such as D'Agents), provide a modified Java VM in order to be able to capture the thread's stack information. This situation is never the case in the commercial Java-based MA systems (such as Aglets, Jumping Beans, or Concordia).

Navigation

As depicted in Figure 6, there are two main approaches concerning the semantics of the navigation operation: statefull navigation (Telescript and D'Agents), and stateless navigation (Aglets, ffMain, and AgentSpace). However, there is an important difference between navigation in Aglets and AgentSpace. In Aglets, the same callback (concretely the `run` callback) is called after the navigation operation. In AgentSpace, the programmer may specify (as an argument to the `moveTo` method) which callback should be invoked after the navigation operation, providing a more elegant and flexible programming solution.

Communication

All systems provide low-level communication primitives, based on the exchange of messages. One aspect analyzed concerns the semantics of communication primitives. Aglets and AgentSpace provide a versatile set of communication primitives such as support for synchronous, asynchronous, and future-based models.

Regarding locality, all MA systems provide inter-agent communication independently of their relative positioning. However, there is an important difference among them: systems like ffMain and D'Agents provide indirect interaction, while the other systems provide direct interaction capabilities. (Of course, D'Agents or ffMain agents can communicate directly using low-level communication mechanisms, such as sockets or pipes. In addition, D'Agents has support for RPC in the Tcl case.) Indirect interaction represents a more general, language independent, and even secure communication model. However, direct interaction is more suitable in property/closed applications mainly due to performance.

Finally, concerning the open channel problem among systems, only AgentSpace and D'Agents support this feature.

Access to external resources

All MA systems provide functionalities and APIs that permit agents to access external resources such as databases systems, and file systems. However, there are some differences between the systems. For instance, in Telescript the mobile agents cannot access external resources directly; only through Telescript places (that are, as seen before, stationary agents regarding this reference model). In D'Agents, Aglets and AgentSpace there is the possibility to prevent foreign mobile agents from accessing critical resources. This is usually controlled by SecurityManagers defined and configured at each agent server level.

Security

From Table 2 it is clear that D'Agents and Telescript are the most secure MA system. In general, every system provides some security features, at least to prevent attacks from foreign mobile agents. This is considered the minimum security level and is provided for example by Aglets. Additionally, other security services should be considered in real and critical applications, such as access control based on user authentication, agent integrity and privacy, and prevention against resources consumption. This last requirement is one of the main security limitations of current Java MA systems due to the fact that the Java VM does not offer any support to control the resources consumed by a Java object.

Web integration

Except for Telescript, the majority of current MA systems provide a strong integration with the Web/Internet technology.

Even with the HTTP protocol extension that occurred in the ffMain system, the majority of current MA systems don't change, or extend, any standard component of the Web (such as the Web server, client, or protocol). Usually, Web integration is performed at the client level through some Java applet that can access an agent server and, consequently, can interact with the agents. Some applets provide a mini-context to run agents in the Web client context (as in the Aglets) while others just provide support for interaction with remote agents (as in the AgentSpace).

8. Summary

In this paper, we have proposed a reference model for mobile agent systems that includes the specification of a set of concepts with the following main goals:

- Clarification of terminology, visions, concepts and their associations.
- Identification of the main components necessary to the support, develop, and manage agent-based applications.
- Analysis, comparison, and evaluation of mobile agent systems.

The issues analyzed included a generic architecture for a MA system and features such as agent execution, management of agent types, generation and management of identities, persistence, navigation, communication, access to external resources, and security.

Finally, and based on the proposed reference model, we compared a set of mobile agent systems representing diverse classes of systems.

This work is not a "finished" reference model. It only has the merit to specify and to compare, for the first time, the state of art of current mobile agent systems, describing their strengths and weaknesses. The mobile agent research community will need to continue to enhance and refine the reference model in the future.

References

[AF89] Y. Artsy, R. Finkel. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer*, September 1989.

[AJDS96] M.P. Atkinson, M. Jordan, L. Daynès and S. Spence. Design issues for Persistent Java: A type-safe, object-oriented, orthogonally persistent system. *Proceedings of The Seventh International Workshop on Persistent Object Systems*. Morgan Kaufmann Publishers, 1996.

[Alb+92] P. Albitz et al. *DNS & BIND*. O'Reilly &Associates Inc., 1992.

[BTV96] J. Baumann, C. Tschudin, J. Vitek (editors). *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems*. Dpunkt, 1996.

[CHK96] D. Chess, C. Harrison, A. Kershenbaum. Mobile Agents: Are they a good idea? In [*VT96*].

[CKTV96] P. Ciancarini, A. Knoche, R. Tolksdorf, F. Vitali. PageSpace: An Architecture to Coordinate Distribute Applications on the Web. *Computer Networks and ISDN Systems*, 28(7-11), 1996.

[Dar98] Darmouth College. *D'Agents: Mobile Agents at Darmouth College*, 1998. http://www.cs.dartmouth.edu/~agent/

[EE98] G. Eddon, H. Eddon. *Inside Distributed COM*. Microsoft Press, 1998.

[FB97] Warwick Ford, Michael Baum. *Secure Electronic Commerce*. Prentice Hall PTR, 1997.

[FKK96] O. Freier, P. Karlton, P. Kocher. *The SSL Protocol, Version 3.0*. Internet Draft, Netscape Communications, March 1996.

[FTP96] FTP Software. *CyberAgents*. 1996. http://www.ftp.com/cyberagents (not available anymore)

[Gel85] D. Gelernter. Generative Communications in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.

[GM96a] General Magic, Inc. *Telescript Technology: An Introduction to the Language*. 1996.

[GM96b] General Magic. *TabizWare*, 1995.
http://www.genmagic.com/Tabriz/

[GM97] General Magic, Inc. *Odyssey Product Information*. 1997.
http://www.genmagic.com/agents/odyssey.html (not available anymore)

[GK94] M. Genesereth, S. Ketchpel. Software Agents. In [*Rie94*].

[Gra95] R. Gray. AgentTcl: a Transportable Agent System. *Proceedings of the CIKM Workshop on Intelligent Information Agents*, (CIKM'95), 1995.

[IBM97] IBM Aglets Software Development Kit - Home Page. Tokyo Research Laboratory, Japan, 1997.
http://www.trl.ibm.co.jp/aglets/

[IKV98] IKV++ GmbH. *Grasshoper, An Intelligent Mobile Agent Platform written in 100% Pure Java*, 1998.

[JRS95] D. Johansen, R. van Renesse, F. Schneider. Operating System Support for Mobile Agents.Proceedings of the 5th Workshop on Hot Topics in Operating Systems., 1995.

[JSW98] N. Jennings, K. Sycara, M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1), Kluwer Academic Press, 1998.

[LDD95] A. Lingnau, O. Drobnik, P. Domel. An HTTP-Based Infrastructure for Mobile Agents. *WWW Journal* (Fourth International WWW Conference), W3C, December 1995.

[LF97] Y. Labrou, T. Finn. A Proposal for a new KQML Specification. *Technical Report CS-97-03*, 1997.
http://www.cs.umbc.edu/kqml/

[Lis88] B. Liskov. Distributed Programming in Argus. *Communications of the ACM*, 31(3), March 1988.

[LO98] D. Lange, M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley. 1998.

[Loc94] H. Lockhart. *OSF DCE Guide to Developing Distributed Applications*. Mc Graw-Hill, 1994.

[MFC98] D. Milojicic, W. laForge, D. Chauhan. Mobile Objects and Agents, Design, Implementation and Lessons Learned. *Distributed Systems Engineering*, IEE, 5 (1988). Also appeared in the *Proceedings of the Fourth USENIX Conference on Object-Oriented Technologies and Systems* (COOTS '98), April, 1998, Santa Fe, New Mexico.

[MS97] M. Mira da Silva, A. Silva. Insisting on Persistent Mobile Agent Systems with an Example Application Area. *Proceedings of the First International Workshop on Mobile Agents*, 1997.

[Mul93] S. Mullender (editor). *Distributed Systems*. (2nd edition), ACM Press/Addison-Wesley, 1993.

[NC98] P. Nixon, V. Cahill (editors). Special Issue on Mobile Computing. *IEEE Internet Computing*, 2(1), 1998.

[Nwa96] H. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3), Cambridge University Press, 1996.

[PL97] A. Park, S. Leuker. A Multi-Agent Architecture Supporting Services Accesses. In [*RP97*].

[PS97] H. Peine, T. Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In [*RP97*].

[Rie94] D. Riecken (editor). Special Issue: Intelligent Agents. *Communications of the ACM*, 37(7), July 1994.

[RP97] K. Rothermel, R. Popescu-Zeletin (editores). *Lecture Notes in Computer Science 1219* (*Mobile Agents'97*) Springer, 1997.

[RSA78] R. Rivest, A. Shamir, L. Adelman. A method for obtaining digital structures and public-key cryptosystems. *Communications of the ACM*, 21(2), 1978.

[SBH96] M. Strasser, J. Baumann and F. Hohl. Mole: A Java-Based Mobile Object System. In [*BTV96*].

[Sch95] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (second edition). Wiley & Sons, 1995.

[Sil99] A. Rodrigues Silva. *Agentes de Software na Internet*. Edições Centro Atlântico. (in Portuguese) 1999.

[Sin98] M. P. Singh. Agent Communication Languages: Rethinking the Principles. *IEEE Computer,* 31(12), December 1988.

[SMD97] A. Silva, M. Mira da Silva, J. Delgado A Survey of Web Information Systems. In *Proceedings of the WebNet'97 – World Conference of the WWW, Internet & Intranet* (Toronto, Canada), November 1997.

[SD98] A. Silva, J. Delgado. The Agent Pattern: A Design Pattern for Dynamic and Distributed Applications. *Proceedings of the EuroPLoP'98, European Conference on Pattern Languages of Programming Systems*, (Irsee, Germany) 1998.

[SMD98] A. Silva, M. Mira da Silva, J. Delgado. AgentSpace: An Implementation of a Next-Generation Mobile Agent System. Proceedings of the Mobile Agents'98. *Lecture Notes in Computer Science 1477.* Springler Verlag. (Stuttgart, Germany) 1998.

[Sun97] Sun Microsystems, Inc., JavaSoft. *Java Remote Method Invocation (RMI)*. 1997. http://www.javasoft.com/products/jdk/rmi

[Sun98] Sun Microsystems, Inc., *The Java Development Kit (JDK),* 1998. http://java.sun.com/jdk/

[VT96] J. Vitek, and C. Tschudin (editors). Mobile Object Systems – Towards the Programmable Internet*. Lecture Notes in Computer Science, 1222,* Springer, July 1996.

[W3C98] World Wide Web Consortium. *Extensible Markup Language (XML)*. 1998 http://www.w3c.org/XML/

[WFN90] E. Walker, R. Floyd, P. Neves. Asynchronous Remote Operation Execution in Distributed Systems. *Proceedings of the 10th International Conference on Distributed Computing Systems*. IEEE, 1990.

[Whi94] J. White. *Telescript Technology: The Foundation for the Electronic Marketplace*. General Magic. 1994.

[Whi97] J. White. Telescript Technology: An Introduction to the Language. *White Paper*. General Magic, Inc. Appeared in J. Bradshaw, Software Agents, AAAI/MIT Press. 1997.

[WPW+97] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M.Young, B. Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. *Proceedings of the First International Workshop on Mobile Agents*. 1997.

[Zim94] P. Zimmermann. *PGP User's Guide (Vol. I e II),* October 1994.

[AdA99] Ad Astra Engineering Inc. *Jumping Beans White Paper,* October 1999. http://www.JumpingBeans.com/

[Obj99] ObjectSpace Inc. The ObjectSpace Voyager Universal ORB. 1999. http://www.objectspace.com/

[ME99] Mitsubishi Electric ITA. Concordia. 1999. http://www.meitca.com/HSL/Projects/Concordia/

[Mil98] D. Milojicic, et al. MASIF, The OMG Mobile Agent System Interoperability Facility. *Proceedings of the Second International Workshop on Mobile Agents*. September 1998. Also appeared in Springer *Journal on Personal Technologies*, 2:117-129, 1998.

[MAL99] *The Mobile Agent List*. University of Stuttgart, Germany. 1999. http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html

[ACT99] Agent Construction Tools (Reticular Systems, Inc.). 1999. http://www.agentbuilder.com/AgentTools/

[FIPA97] Foundation for Inteligent Physical Agents (FIPA). *FIPA 97 Specification, Version 2.0, Part 2, Agent Communication Language.* October, 1998.http://drogo.cselt.stet.it/fipa/spec/fipa97/fipa97.htm

[Wel90] B. Welch. *Naming, State Management and User-Level Extensions in the Sprite Distributed File System*. Ph.D. Thesis, Technical Report UCB/CSD 90/567, CSD (EECS), University of California, Berkeley. April 1990.

[Sha86] M. Shapiro. Structure and Encapsulation in Distributed Systems: The PROXY Principle. *Proceedings of the 6th International Conference on Distributed Computing Systems*. May 1986.

[SDP92] M. Shapiro, P. Dickman, D. Plainfossi. Robust, Distributed References and Acyclic Garbage Collection. *Proceedings of the Symposium on Principles of Distributed Computing*. August 1992.

[DA98] T. Dierks, and C. Allen. The TLS Protocol, Version 1.0. *Internet RFC 2246*, January 1998.

[Mor99] J. P. Morgenthal. ObjectSpace Voyager Core Package Technical Overview. In [*MDW99*].

[MDW99] D. Milojicic, F. Douglis, R. Wheeler (editores). *Mobility: Processes, Computers, and Agents*. Addison-Wesley / ACM Press. February, 1999.

Appendix - Guide of Agent Electronic Resources

This appendix compiles a significant set of electronic references related to the Internet, mobile and software agents. This compilation is organized on the following groups of topics:

- *Organizations*: Groups of enterprises and research laboratories with the main goal to promote agent technologies and agent paradigm, as well as to promote the discussion and creation of standards in this area.
- *Introductory Readings*: Set of classic references, simple enough to provide an introductory and general understanding of the paradigm, applicability and usability of software agents and, in particular, of mobile agents.
- *Agent Catalogues*. Excellent resources to start looking for general information about mobile agents. Agent catalogues contain a significant number of electronic resources to different places organized by different criteria. Some of these resources contain, in addition, electronic newsletters and support for discussion groups.
- *Specifications*. Public documents in draft or final result developed by specialized workgroups in the context of some organizations.
- *Mobile Agent Systems and Projects*. Electronic references to the majority of current MA systems and/or projects.

Organizations

Title	URL
Agent Society	http://www.agent.org
Foundation for Intelligent Physical Agents (FIPA)	http://drogo.cselt.stet.it/fipa/index.htm
Object Management Group (OMG)	http://www.omg.org
AgentLink	http://www.agentlink.org
Climate	http://www.fokus.de/cc/ima/climate
Agent Patterns	http://www.scs.carleton.ca/~deugo/Patterns/Agent

Introductory Readings

Title	URL
A hands-on look at Java Mobile Agents (Joseph Kinry, Daniel Zimmerman)	http://computer.org/internet/ic1997/w4021abs.htm
Agents to roam the Internet (George Lawton)	http://www.sunworld.com/swol-10-1996/swol-10-agent.html

An introduction to agents (Todd Sundsted)	http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html
Agents on the move (Todd Sundsted)	http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto.html
Mobile Agents: Are they a good idea? (Colin G. Harrison et al.)	http://www.infosys.tuwien.ac.at/Research/Agents/archive/special/mobagtibm.ps.gz
Mobile Agents: Explanations and Examples (William R. Cockayne, Michael Zyda)	http://www.manning.com/Cockayne/index.html
Publications (Software Agents Group)	http://agents.www.media.mit.edu/groups/agents/publications
Mobile Agents White Paper (General Magic)	http://www.genmagic.com/technology/techwhitepaper.html

Agent Catalogues

Title	URL
UMBC AgentWeb	http://www.cs.umbc.edu/agents
Mobile Code, Agents and Java (Distributed Systems Group)	http://www.infosys.tuwien.ac.at/Research/Agents
Cetus Links: Distributed Objects & Components: Mobile Agents	http://www.rhein-neckar.de/~cetus/oo_mobile_agents.html
The Mobile Agent List (Univ. Stuttgart)	http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html
Agent Construction Tools (Reticular Systems, Inc.)	http://www.agentbuilder.com/AgentTools/
OpenSesame's Agent Links	http://www.opensesame.com/agents
Agents (JavaWorld)	http://www.javaworld.com/javaworld/topicalindex/jw-ti-agents.html
The @gency (Serge Stinckwich)	http://www.info.unicaen.fr/~serge/sma.html
Agent Technologies (Centre for Advanced Learning Technologies)	http://www.insead.fr/CALT/Encyclopedia/ComputerSciences/Agents
Intelligent Agents (CompInfo)	http://www.compinfo.co.uk/tpagnt.htm
Intelligent Software Agents (Sverker Janson)	http://www.sics.se/isl/abc/survey.html
Mobile Code Systems (W3C)	http://www.w3.org/MobileCode
Agent References (Distributed Systems Group)	http://www.infosys.tuwien.ac.at/Research/Agents/ref.html

Specifications

Title	URL
OMG's MASIF: Mobile Agent Facility Specification (General Magic et al.)	http://www.infosys.tuwien.ac.at/Research/Agents/archive/special/mafdraft7.ps.gz
FIPA 97 specification (FIPA)	http://drogo.cselt.stet.it/fipa/spec/fipa97.htm
FIPA 98 specification (FIPA)	http://drogo.cselt.stet.it/fipa/spec/fipa98/fipa98.htm

Mobile Agent Systems and Projects

Title	URL
Follow Me (EU Project)	http://hyperwav.fast.de/FollowMe
JIAC - Java Intelligent Agent Componentware (DAI-Lab)	http://dai.cs.tu-berlin.de/english/forschung/projekte/JIAC
OnTheMove (ACTS Project)	http://www.sics.se/~onthemove
Agent Building Environment (IBM)	http://www.networking.ibm.com/iag/iagsoft.htm
AgentSpace (IST/INESC)	http://berlin.inesc.pt/agentspace/
Aglets SDK	http://www.trl.ibm.co.jp/aglets
Ajanta (University of Minnesota)	http://www.cs.umn.edu

AMETAS (Michael Zapf)	http://www.vsb.cs.uni-frankfurt.de/~zapf/project.html
ARA - Agents for Remote Action (University of Kaiserslautern)	http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html
Caltech Infospheres Project (Infospheres Group)	http://www.infospheres.caltech.edu
Concordia (Mitsubishi Electric Information Technology Center America)	http://www.concordia.mea.com
D'Agents (Agent Tcl) (Dartmouth College)	http://www.cs.dartmouth.edu/~agent
ffMAIN - The Frankfurt Mobile Agents Infrastructure	Http://www.tm.informatik.uni-frankfurt.de/ma/ffmain.html
GrassHopper	Http://www.ikv.de/products/grasshopper
JAFMAS	Http://www.ececs.uc.edu/~abaker/JAFMAS
Java Agent Template (H. Robert Frost)	Http://cdr.stanford.edu/people/frost-bio.html
JATLite (Stanford University)	Http://java.stanford.edu
Messengers (University of California)	Http://www.ics.uci.edu/~bic/messengers
MOA - Mobile Objects and Agents (OpenGroup)	Http://www.camb.opengroup.org/RI/java/moa/index.htm
Mole (University of Stuttgart)	Http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html
Plangent (Toshiba Corporation)	Http://www2.toshiba.co.jp/plangent/index.htm
TACOMA - Troms&oslash; And COrnell Moving Agents	Http://www.tacoma.cs.uit.no/
AgentBuilder (Reticular Systems, Inc.)	Http://www.agentbuilder.com
Jumping Beans	http://www.JumpingBeans.com/
Voyager (ObjectSpace)	http://www.objectspace.com/