Improving Current Agent Support Systems:Focus on the Agent Execution System

Alberto Silva

INESC / Tecnical University of Lisbon

Abstract

We propose in this paper a generic architecture to support, develop, manage, and interact with agent-based applications. In this context, we detail two main components of the architecture, namely: the AES (Agent Execution System) and the AEE (Agent Execution Environment). For both of them we describetheir main concepts, data structures and interfaces.We also propose a set of characteristics that we believe will also be present in the next-generation AES, such as: independence between AES and AEE, i.e., support to multi-language agents; execution placeshierarchically organized; interaction mechanisms amongst AESs and agents based on the event drivenmodel with potentially abortable events; and management and administration AESs capabilities

1 Introduction

Nowadays, there are many ASS (mobile Agent Support Systems) proposals – such as Aglets [IBM97],Agent-Tcl [Gray95], Odysseya [GM97], Tacoma [JRS95] – that present a common set of functionalities and purposes. Nevertheless, they also present some important technical and even conceptual differencesamongst themselves.

For instance, Telescript [Whi96] was the usual reference for ASS, with a very high technological level.However it is/was a proprietary system, with a difficult programming language, and not well adapted to adynamic and open environment such as the Internet.

On the other hand, a system like the HTTP-based agent infrastructure [LDD95] is language and systemindependent, but shows severe limitations of the overall performance and difficulties in developing complexapplications.

The Aglets Workbench, that represents Java-based ASSs, lacks an elaborate object model (e.g., without thenotion of execution places hierarchically organized; without management operation on agent families andclusters) and technical capabilities (e.g., just two access control levels; without the notion of an agent-classes manager).

We expect that in the next few years these ASSs would be progressively improved in order to support thedevelopment and execution of more flexible, reliable, secure and efficient agent-based applications (ABA).In order to achieve

that, we need better ASSs that incorporate the best features of existing ASS. For example, they should include the independence of HTTP-based ASS, the technology for migrating agents asfound in Telescript, and the integration with Java as supported by the Aglets workbench.

In this paper we give a very small contribution on that direction by proposing a complete, integratedconceptual framework for a next-generation ASS. Instead of being a final product, the paper intends only tocontribute to the definition of the future ASS's main characteristics. The focus of this paper is mainly onthe ASS aspects. For a general overview of software agents and ABA the reader may consult a very widerelated literature [GK95,BTV96,SMdSD97].

The paper is organized as follows. The next section presents in overview of the global architecture proposedfor a next-generation ASS. We will then focus on the most important components: the agent executionenvironment in section 3, and the agent execution system in section 4. In section 5 we summarize thecontributions with a small discussion in the Internet and Intranet contexts.

2 Global Architecture Overview

Our ASS proposal is divided into three major components: AES, for Agent Execution System; ACS, for Agent Class System; and AEE, for Agent Execution Environment .

In the lower level of the architecture there are the systems that support the storage and access to agentclasses and their adequate execution.

- AES (Agent Execution System) – provides support to agent navigation and communicationoperations; provides persistence support; security; etc. The AES keeps its own database (or objectstore) where it tracks pertinent information about all agent and execution places (EP) supported, as wellas data structures to support inter-agent communication. Additionally, the AES provides a set of communication protocols with similar and different AESs types.

- ACS (Agent Class System) – keeps (in its own internal database) a dynamic set of agent classes and provides a consistent and secure access interface, in order that other components (such as: M-ACS,M-ABA, DE-ABA, and even agents) may reference and instantiate, at run-time, a given set of classes.

- AEE (Agent Execution Environment) – provides an execution environment, corresponding typically tothe interpreter or virtual machine attached to the programming language in which the agents weredefined.

- DE-ABA (ABA Development Environment) – supports the development, test and debug of agentclasses and more generically of ABA. Ideally DE-ABAs should provide a powerful, complete andintegrated environment, with several paradigms and techniques such as those found in the mostmodern tools (e.g., Visual Basic, Delphi, PowerBuilder, J++, etc.)

2.1 AEE – Agent Execution Environment

The AEE is a virtual machine (VM) that allows the agent execution. With a given set of APIs (dynamic link libraries, packages, etc.) AEEs allow agents to access functionalities provided by AESs, ACS, or evendifferent external resources. There is a strong relationship between the AES and the AEE, due to the fact thefirst restrict in fact the agent execution to the context of the AEE. There is a possible variant of the conceptual model depicted on Figure 1 that is illustrated in Figure 3. Thisvariant happens when the AES is itself supported by a VM (e.g., Java VM). In this case, and when theagents are executed by the same VM, the agents can be executed in the internal computational context of theAES as a thread of execution. In this case, the AES exported interface is comparatively easier to implementand more efficient, because it is based on direct method invocations within the AES/AEE process. We call"solution 1" to this concrete implementation of the proposed architecture.
On the other hand, the general solution requires a physical separation between the AES and AEE/agent processes. This is required, for example, to support several agents implemented in multiple programminglanguages. We call this variant of the basic architecture "solution 2". Although the interface AEE/AES issimilar for both solutions, the implementation of solution 2 is harder to build and less efficient then itscounterpart for solution 1.It should be noted that any language can be used for building agents – not only Java – as far as the AES provides an equivalent API implementation. (However, we still prefer a simple, object-oriented, type-safe programming language for obvious reasons).

3 AES – Agent Execution System

The AES maintains a private data repository about EP (execution places), agents, end-users, groups, andtheir respective permissions.
The AES is typically a process executing multiple threads. Its main goal is to provide facilities that transport,store, manage, control and communicate with and between agents. As we mentioned in the previoussection, the AES should provide the same capabilities whether the agent is being executed internally or externally to the AES context.
The EP (execution place) class has mainly two objectives. First, to provide a conceptual and programmingmetaphor where agents are executed and meet other agents. Second, to provide a consistent way to defineand control access levels, and to control computational resources.The EP has a unique global identification (based on its AES's URL) and knows the identification of itsuser/manager. It also maintains a keyword/property list that allows an informal characterization. Optionally,the EPs can be hierarchically organized.
The EP can also contain the maximum and current number of agentsallowed in order to support elaborate resource management.Each AES has at least one EP – called "default EP" – that is used every time an EP is not known or not evenspecified.

During its lifetime, a mobile agent can be executed in several EPs (either local or remote). An agent is visitingan EP when it is not executing in the EP where it was created originally. In order to keep track of its agents,

5the EP keeps a list with its visitant agents and another with its native agents. The EP also knows in which EPits native agents are executing at a given point of time.The MetaAgent class keeps agent-related information, namely: its own identification, its native EPidentification; its end-user identification; the maximum and current number of created agents and EPs; andthe identification of its agent class. Based on the agent class information, it is then possible to get all themessages (calling getMsgs) and public methods (getInterfaces) that the agent knows how to handle.Additionally the MetaAgent class keeps the following information: the current EP where the agent is beingexecuted; a list of received messages not yet handled by the agent; a flag telling if the agent is beingexecuted internally (as a thread) or externally (as a process) regarding its current AES; and the AES internalidentification.

When it is executing in their original AES, the agent has just one MetaAgent instance. However, when it isvisiting another EP it has two MetaAgent instances – one in its native EP and the other in the currentlyvisited EP. The instance in the visited EP provides a consistent and efficient access and management-mechanism for the agent. The instance in the native EP helps to solve the "open channels" issue and provides local access and management capabilities. There should also exist a replication mechanism to keepthe consistence between the original (kept in the native EP) and the temporary instance (kept in the visitedEP) – for instance, based on the centralized proxy model.Agents usually interact indirectly by exchanging messages via their MetaAgent instances (either local or remote).

The AES maintains lists of users and groups to implement the permission and control access mechanism. Auser may belong to one or more groups. Groups may be hierarchically organized to simplify permissions.This means that all users of some specialized group have implicitly all the permissions they inherit from themore general groups, although the contrary is obviously not true.

4   Interface with the AEE

The AEE interface is the main programming interface, to be used by agents and agent aplications alike.Agent applications use this interface to create, duplicate, and transfer agents. It can also be used tocommunicate between agent applications themselves.

5   Interface with Management Tools

The AES is configured, managed, and generally maintained by management tools that are just specializedagent-based applications. Typical operations include those involving.

- EP (native agents, visiting agents, and so on);

- users and user groups;

- permissions (relationships between EP and user groups).

These operations should be extended to remote AES so that a number of AES can all be managed centrally by only one system administrator.Using these tools, an "agent system administrator" (that is, a real person) can easily have an overall pictureof what is going on with agents under his or her responsibility and take actions as required. For example, itwill be needed to start local and remote AES, create users and user groups, set-up and change permissions,create and stop agents, and so on.

Finally, the AES administrator will also need to manage the database, the local file system and potentiallyalso the communication system. These will probably reside outside the AES, so there is a strong interaction between the operating system and the AES which needs to be better understood.

The management tools themselves can be implemented by agents or use any other technology. However,implementing them with agents will help users trust the AES and provide useful feedback for the AES provider as well.

6   Interface with DBMS

This interface to the DBMS enables the AES to save its internal data structures – agents, messages, and soon – persistently, reliably and safely.

This interface can be implemented in one of two basic approaches: either the AES keeps its own store withits own interface (represented by "1" in figure 5); or uses an existing DBMS via some standard API such asODBC or JDBC.

7   Summary and Discussion

In this paper we proposed a general architecture for an agent support system with several interestingfeatures. These include.

- the execution place as an execution and meeting point for agents, but also as a control mechanism for checking authorizations, access control, and resource management;

- users and user groups with permissions that are associated with execution places;

- support for one or multiple agent programming languages that permit application programmers toachieve the right balance between flexibility and efficiency;

- internally and externally executed agents, again permitting the right level of security and efficiencydepending on the application characteristics;

- communication protocol between agents and their support system based on pre-defined methods andcallbacks in both directions;

- a flexible but still simple event model that gives agents some control about themselves;

- a core set of classes and interfaces for accessing and managing agents and agent system.

Although quite straightforward, the architecture proposed in this paper is novel in several aspects. On onehand, it is a general framework not biased by any particular programming language as many other architecture proposals. On the other hand, it includes not only components for the agent system itself butalso many other components that will be needed to manage and utilize the agent system.In the future, we believe current agent systems will evolve to support many of the features proposed in this paper. However, before achieving that, a number of pragmatic decisions will have to be taken:

- Java and its libraries (RMI, JDBC, security, and so on) will be even more used for supporting not onlyagents but also agent support systems;

- many agent systems will support more than one agent programming language to take advantage of existing source code; URL will continue to be used to identify agents and agent systems, even though it depends on the physical machine address (although agents can already be identified by any other mechanism);

- persistence will be based on external relational databases or the file system (with object serialization)even though agents require much more elaborate persistent support [MdSA96,MdS96,MdSS97];

- there will be many different communication mechanisms between agents, including synchronous andasynchronous, local and remote, direct and indirect, point-to-point and multicast, and so on;

- there will be many external interfaces to access resources not supported by the AES;

- there will be many other external functionalities (for example, access control) that programmers willhave to write because existing ones cannot be integrated with an agent system;

The new JDK 1.1 for Java – with its libraries for communication, persistence and database access – providesan extensive support to build AES. (RMI is almost an agent system by itself!) However, each AES will useJava in its own way, and as a result a number of different, incompatible AES will appear. The new MobileAgent Facility [Cry+97] from OMG can help to make different AES communicate, but it does nothing tonormalise the AES itself.

If each AES offers its own programming model, then users will not be confident

on agent technology andwill have many troubles using a particular AES. It is an objective of this paper to help providers of agentsystem to build simple, integrated AES that can be easily understood and used by normal application programmers. In order to achieve that, we proposed a single, general, complete architecture for all AES.

8 References

J. Baumann, C. Tschudin, J. Vitek, editors. Proceedings of the 2 nd ECOOP Workshopon Mobile Object Systems (Linz, Austria) . Dpunkt, 1996.

Crystaliz Inc. et al. Mobile Agent Facility Specification (Joint Submission) - Draft 5.April, 1997.

D. Hohansen, R. Renesse, F. Schneider. An Introduction to the TACOMA Distributed System . Computer Science Technical Report 95-23. University of Tromso, Norway,1995.

A. Lingnau, O. Drobnik, P. Domel. An HTTP-Based Infrastructure for Mobile Agents.In Proceedings of the Fourth Int'l WWW Conference , 1995.

M. Mira da Silva, M. Atkinson. Combining mobile agents with persistent systems:Oportunities and challanges. In Baumann et al. [BTV96].

9 Picture of Tree

Tree Data Structure

Root Node / Internal Node

Internal Node

Leaf Node
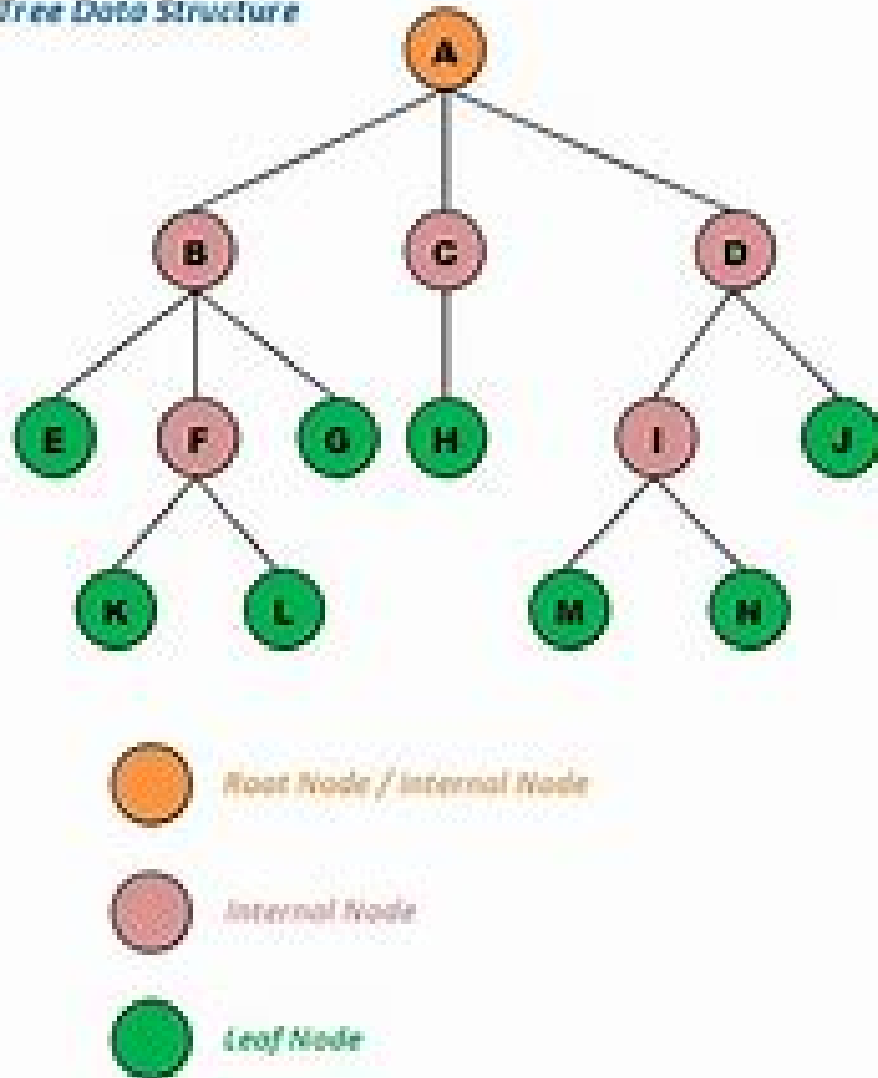
10   Table of Student

SL.	Name
1.	Rifat
2.	Jahangir
3.	Shoriful
4.	Sajib
5.	Samapan

11   Table of Non-Primitive Data Stucture

Linear	Non-Linear
Array	Graph
Stack	Tree
Linked List	
Queue	

12   Mathmatical Equation

The equation is

$$E = mc^2$$

$$y = mx + c \tag{1}$$

$$\sum_{i=1}^{\infty}\prod_{i=1}^{\infty} \tag{2}$$

Limit

$$\lim_{a \to 4} \frac{a+3}{a+2} \tag{3}$$

Value of A:

$$a = 8 + 3 - 2$$
$$= 11 - 2 \tag{4}$$
$$a = 9$$

Matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{5}$$