GOING BEYOND MOBILE AGENT PLATFORMS: COMPONENT-BASED DEVELOPMENT OF MOBILE AGENT SYSTEMS

PAULO JORGE MARQUES, LUÍS MOURA SILVA, JOÃO GABRIEL SILVA
CISUC, University of Coimbra, Portugal

{pmarques, luis, jgabriel}@dei.uc.pt

ABSTRACT

Although mobile agents are a promising programming paradigm, the actual deployment of this technology in real applications has been far away from what the researchers were expecting. One important reason for this is the fact that in the current mobile agent frameworks it is quite difficult to develop applications without having to center them on the mobile agents and on the agent platforms. In this paper, we present a component-based framework that enables ordinary applications to use mobile agents in an easy and flexible way. By using this approach, applications can be developed using current object-oriented approaches and become able of sending and receiving agents by the simple drag-and-drop of mobility components. The framework was implemented using the JavaBeans component model and provides integration with ActiveX, which allows applications to be written in a wide variety of programming languages. By using this framework, the development of applications that can make use of mobile agents is greatly simplified, which can contribute to a wider spreading of the mobile agent technology.

Keyworks: Mobile Agents, Software Components, Reusability, Internet Computing, Distributed Systems

1. INTRODUCTION

Mobile agents (MAs) are an exciting new programming paradigm. They can be applied with significant advantages in many domains like network management, information filtering, mobile computing and electronic commerce. Systems that implement mobile agents have now been around for some years and many agent platforms are available from the research community and industry [1-4]. Nevertheless, the paradigm has been lacking from applications running on a production environment and has gained the reputation of being "a solution looking for a problem" [5]. The actual delivery of the technology has fallen short on the initial expectations of the researchers.

We believe that one important factor preventing the widespread of mobile agents is the lack of proper support for application development. Research has been mainly focused on the agent technology and on mobility issues rather than on the support needed for real-world application development. This issue has two aspects: the user and the developer.

From the viewpoint of the user, if an application makes use of mobile agents then it is necessary to install an agent platform. The security permissions given to the incoming agents must be configured and the necessary hooks to allow the communication between the agents and the application have to be setup. While some of these tasks can be automated using installation scripts, this entire setup package is too much of a burden for the average user. Usually, the user is not concerned with mobile agents nor wants to configure and manage mobile agent platforms. The user is much more concerned with the applications than with the middleware used in the background. In the currently available mobile agent systems, the agents are central and widely visible. They are not the background middleware but the foreground applications.

From the point of view of the programmer, constructing an application based on mobile agents is a complicated process. Many times the applications themselves have to be coded as special type of agents – *stationary agents*. In other cases, stationary agents have to be written to interface the applications with the incoming agents. Although the mobile agent concept is a useful structuring primitive, nowadays mobile-agent frameworks require that applications be developed in different ways from current practices. This prevents programmers from easily embracing a useful programming concept, and from benefiting of established advantages of using mobile agents: network bandwidth reduction, encapsulation of protocols and asynchronous remote execution, among others [1].

In this paper, we present a component-based approach for easily supporting the mobile-agent concept when developing applications. In this approach there is no agent platform. The applications become capable of sending, receiving and interacting with mobile agents by using well-defined software components [6]. Applications are developed using the current industry best-practice software methods, and can become agent-enabled by using mobility components. If a Visual Development Environment is used, this can be accomplished by the simple drag-and-drop of components from a component palette and by visually configuring their properties and interconnections. This is a step forward over the traditional development approaches used in the platform-based MA systems. Because in this approach there is no mobile-agent platform and because the emphasis is put on the application and not on the agents, we call this approach *ACMAS – Application-Centric Mobile Agent System*.

The rest of this paper is organized as follows. Section 2 details the ACMAS approach to application development. Section 3 presents the implementation of the paradigm. Section 4 discusses the related work. Finally, Section 5 presents the conclusions and future work to be done.

2. THE ACMAS APPROACH

2.1 ENABLING THE APPLICATIONS TO USE MOBILE AGENTS

In the ACMAS approach, the application is central. Mobile agents are just a part of the system playing specific roles. It is not required that all the system is constructed around mobile agents. Components provide a way of closely integrating the mobile agent paradigm with current OO software development approaches.

An application is enabled with the capacity of sending and receiving agents by using components (Figure 1). Each agent arrives and departs from the application that it is specific. The application knows the interface of the agents and the agents know how to interact with the applications.

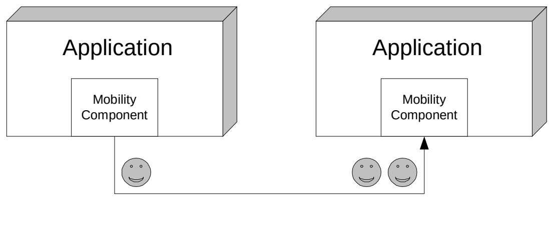

Figure 1 – Applications become agent-enabled by using mobility components.

The programmer is given a package of components that are used for developing agent-enabled applications. Programming those applications is done by the drag-and-drop of components and by configuring their properties and interconnections. All this can be accomplished in a Visual Development Environment.

Each component implements a certain functionality, which is called a *service*. Examples of services include security, persistency, agent tracking, disconnected computing, agent coordination and communication. When developing an application only a subset of these services will be required. Because each service is implemented in a separate component, they can easily be included or removed from an application. Each application only makes use of the functionalities it actually requires. This makes possible to build an application that uses mobile agents in a much more flexible way when compared with the use of a traditional monolithic mobile-agent platform.

The agents interact directly with the application and the application with the agents. Nevertheless, it is necessary to encapsulate this interaction though a simple wrapper that provides security and allows the agents to query and interact with the available services. This is needed because upon the arrival of an agent, it must have a well-defined entry-point that allows it to access and interact with the system in a secure and trustful way. Due to space constraints, we will not discuss the security issues of the approach.

2.2 BENEFITS OF USING A COMPONENT-BASED APPROACH

In the current mobile agent systems, there is always an entity called “Mobile Agent Platform”. The platform is responsible for receiving the agents and for providing certain functionalities. It does not matter if there are many different types of agents interacting with many different types of applications or just one type of agents and one application. All the agents arrive and depart from the platform. In the ACMAS approach, the agents arrive and departure from the applications they have interest in interacting with. Additionally, only the necessary components for supporting the needs of each application are included. This is illustrated in Figure 2.

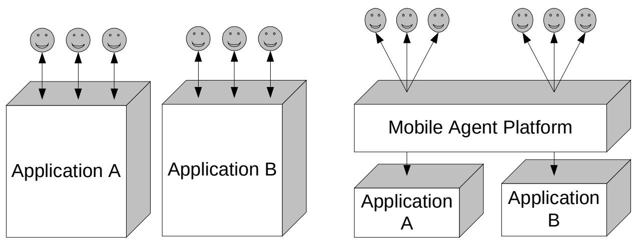

Figure 2 – Platform-based vs. Application-based mobile agent systems.

Using a component-based brings the following benefits:

Flexibility

Each application uses exactly the components necessary to support the mobile-agent requirements of its application domain. Each component can be transparently added or removed from the application. Additionally, any new service can be written in a new component, without interfering with the rest of the existing framework.

Easy Software Development

In terms of software development, the ACMAS approach has clear advantages. The applications are constructed using the best-known practices of software engineering. Introducing mobile agents only involves the drag-and-drop of the necessary components and the configuration of their properties, all done within a Visual Development Environment.

In addition, because the agents interact directly with the applications, there is no need for developing stationary agents or to program the applications as agents.

Fault-Isolation

If an agent platform crashes by some reason, then all the agents and running applications are affected. In ACMAS there is fault-isolation between applications since a crash in one application or in its mobility components does not affect the other running applications or their agents.

Seamless Integration

In ACMAS, the applications do not have to be written around mobile-agents. Mobility components are just like any other middleware being used in an application (e.g. network management protocols like SNMP [7] or distributed computing middleware like RMI [8], JINI [9] and CORBA [10]). This is especially important when it is necessary to integrate mobile agent technology with legacy applications. If an application has already been developed using other middleware, it is still possible to enable it to receive and send agents by including the mobility components into it.

User Acceptance

If a user is given an application developed using the ACMAS approach, the use of mobile agents is not visible. From the user's perspective, the application is just as any other application that he learns how to use. This contrasts with the platform-based approach, where the user is not shielded from the mobile agent middleware.

Being conscious of the existence of mobile agents has an important psychological implication that makes the dissemination of the technology difficult: the user is afraid of installing a platform capable of receiving and executing code without his permission. This happens even though mobile code exists in technologies like Java, in particular in RMI and JINI. The fundamental difference is that in those cases, the user is shielded from the middleware being used.

In many cases, using mobile agents does not introduce an increased security threat, especially if proper authentication mechanisms are in place [11]. However, because the current agent platforms do not shield the user from the middleware, the risk associated with this technology is perceived as being higher, which causes users to back away from applications that make use of mobile agents. ACMAS can contribute to ease this situation.

3. IMPLEMENTATION

In order to validate the idea of Agent-Centric Mobile Agent Systems, we have implemented a framework called MOBILITY. MOBILITY is a package of components written in Java, using the JavaBeans [12] component model. These components give an application the ability to send and receive agents, provide a number of supporting services for the agents and the application, and enable new programmer-defined services to be added in a straightforward manner.

One very important concern while writing MOBILITY was to make it very modular and small footprint. A very flexible plug-in architecture was devised in order to be able to develop any number of services and to seamlessly integrate them in the existing infrastructure.

In this section, we give an overview of the implementation of MOBILITY. In particular, we will examine the Mobility Component and its support of services.

3.1 FRAMEWORK INTEGRATION

The Mobility Component is coupled with the application and with the mobility supporting services through events (Figure 3).

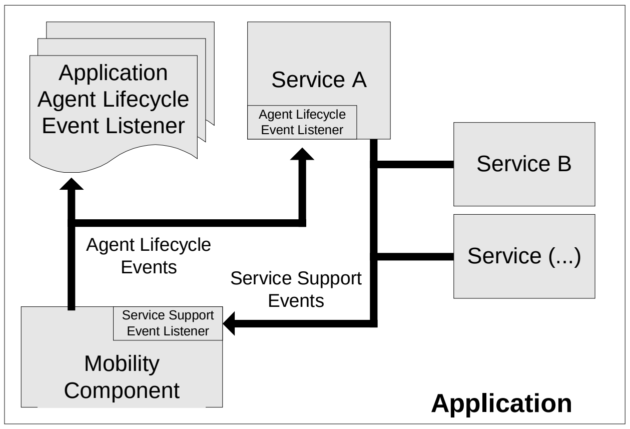

Figure 3 – Interactions between the Mobility Component and the Application.

Two major sets of events exist: *Agent Lifecycle Events* and *Service Support Events*. Agent Lifecycle Events represent changes in the state of a running agent (e.g. when an agent arrives or departs from the application). Service Support Events enable the Mobility Component to incorporate new services at runtime and to be notified if they are no longer available.

Whenever an agent changes its execution state (e.g. arrives, departures or dies), an Agent Lifecycle Event is fired. The application and the mobility services can register their listeners with the Mobility Component, to be notified when one of those events occurs. When an event takes place, the listeners are able to examine the agent responsible for the event and if necessary, call methods on the agent or in other modules of the application. This is especially important for the higher-level services since they can make use of those events to accomplish their functions. For instance, a persistence service can checkpoint an agent to disk after receiving an `OnAgentArrivalEvent` (one of the Agent Lifecycle Events) and remove it after receiving an `AfterAgentMigrationEvent`. For certain events like an agent arrival or an agent migration, each event listener has also the capability of vetoing the event. For instance, a security service may decide not to allow an incoming agent to migrate into the application by vetoing the `OnAgentArrivalEvent`.

Service Support Events exist to allow a service to be incorporated or removed from the system at runtime, and to allow services to be configurable in a Visual Development Environment. A service registers itself with the Mobility Component as a source for those events and, whenever it becomes available, it fires an event that notifies the Mobility Component of the occurrence. Also, if the service whishes to be removed from the list of available services, it fires an event requesting to be removed of the available services list. The existence of this event set is extremely import because if allows development tools like BDK [13] to setup adapters between components. All the programmer has to do is to drag-and-drop the Mobility Component, the service components and interconnect them. This is done without having to write a single line of code.

3.2 INSIDE THE MOBILITY COMPONENT

The Mobility Component is extremely small and supports only the core functionalities needed to migrate and manage agents efficiently. Figure 4 shows the major sub-modules of the component.

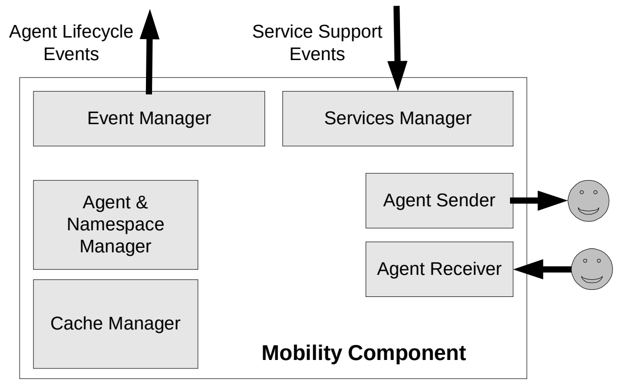

Figure 4 – The Mobility Component.

The *Agent Sender* and *Agent Receiver* modules are responsible for sending and receiving agents. These modules take care of the serialization and deserialization of agents and of making them available for other modules. MOBILITY supports week migration based on a `jump()` instruction. This jump instruction takes a parameter in the form of an URL: "`mob://machine:port/namespace#methodToExecute`". The code of the agents being migrated between applications can be transmitted along with the state of the agent or may be downloaded from URL based sites (e.g. http or ftp servers). These two modules are responsible for implementing those features.

Whenever an agent is arriving, before receiving the code of the agent, the *Cache Manager* module is consulted. The code of an agent is stored inside a Jar file, having each Jar a unique identifier. The *Cache Manager* implements a two level cache, being the first level a memory cache and the second level a disk cache. If the agent is already present in one of the caches, only the state of the agent is transmitted between applications. Upon arrival, if the code of the agent is not present in cache, it is stored. The used policy is a variation of the LRU algorithm that takes into account the size of the code being replaced and the available space in cache.

The *Agent & Namespace Manager* is responsible for managing the running agents. It is the control module inside the component. Within this module, there is an agent table and a namespace table. Each agent exists inside a namespace and is able to move between namespaces. Whenever an agent migrates, it migrates to a namespace. This module is responsible for starting, stopping and monitoring agents. It is also through this module that the namespaces are managed in the system.

The *Event Manager* module is responsible for sending the Agent Lifecycle Events to the registered listeners, whenever an agent changes its state. It is also responsible for ensuring that the vetoing of events takes the desired semantic action (e.g. the vetoing of an `OnAgentArrivalEvent` should prevent an agent from arriving at the application while the vetoing of an `OnAgentMigrationEvent` should prevent an agent from leaving the application).

Finally, the *Services Manager* is responsible for listening to incoming Service Support Events, making higher-level services available or unavailable. Implementing higher-level services implies implementing a `ServiceFactory` capable of creating service object instances on demand. Each one of those instances implements a well-defined `ServiceInterface` that is made available to agents requesting the service. Associated with each service there is also a `ServiceDescriptor` that identifies the service. An agent can request a service by name or by `ServiceDescriptor`. It is also possible to ask which services are registered with the Mobility Component. If an agent wants to use a service for which it does not have the corresponding service interface, it can nevertheless discover at runtime what is available in the service by using the Java introspection mechanism [12]. This is important in cases where there is a newer version of a service that is well known to the agent but, because it is a newer version, the agent does not carry the appropriate binary interface. In this case, the agent can check if a certain method exists and the parameters match, and call it by name.

All the features of the Mobility Component are accessible through properties [12]. Thus, for instance, it is possible to configure incoming and outgoing TCP/IP ports for the component, maximum and minimum cache sizes, if they are used or not, maximum number of running agents, and many other characteristics of the component by simply changing these properties in a Visual Development Environment.

3.3 ACTIVEX SUPPORT

The MOBILITY package was implemented using the Java programming language and using the JavaBeans component model. Nevertheless, we have also decided to support the ActiveX component model, that is based on Microsoft's COM [14].

The Importance of Supporting ActiveX

We have taken this decision because of the wide adoption of this component model in the software industry and in research. We believe that by supporting the ActiveX, a much wider usage base and a much richer set of environments where to use our framework will be available.

Another important point is that any language that has the necessary mechanisms to make use of COM and ActiveX is able to use the mobility components. At the present, most of the languages that available for Windows [15] have that capability. By supporting ActiveX, we are no longer limited to develop the applications in Java. We can program the applications in languages like Visual C++ [16], Visual Basic [17] and Delphi [18]. This happens without having to implement any special layer between our components and the target languages. During our experiments, we developed a simple instant messaging application in Visual C++, Visual Basic and Java. The agents migrated and executed in all the client applications independently of the language where they were written.

Mapping JavaBeans into ActiveX

For supporting ActiveX, we have used Sun's ActiveX/JavaBeans bridge [19]. The bridge provides an encapsulation of the JavaBeans components mapping their properties, events and methods into a neutral form. The ActiveX container that hosts the component is able to access these properties, events and methods as if the component were an ordinary ActiveX control.

The major difficulty associated with using the mobility as ActiveX components is that only simple data types are provided in ActiveX. For instance, when the Mobility Component fires an Agent Lifecycle Event, the `Agent` that caused the event is present in the event information. Because `Agent` does not correspond to any data type supported by ActiveX, that information is passed as a pointer to an `IDispatch` interface [14]. To complicate matters, the bridge does not support the querying of this interface making impossible to use this information.

To solve the problem we have created helper components – *Crackers*, one for each supported event. These components take as input the reference to the object referred by the `IDispatch` pointer and flat the structure of the object into data types that can be understand by the container. The programmer includes the Mobility Component into the application along with the Agent Lifecycle Event Cracker Component. Then, when writing the event handler, the programmer redirects the incoming event passed as parameter into the cracker component. The data available in the event becomes then available as properties on the cracker component.

Because multiple events may be fired before an event handler has the opportunity to process them in the handler, a locking scheme has been implemented in the crackers.

4. RELATED WORK

Examples of mobile agent platforms include Aglets from IBM [1], Concordia from Mitsubishi [4], Grasshopper from IKV++ [3] and Mole from the University of Stuttgart [20]. All these mobile agent systems are platform-based focusing on the issues of mobility, not on the support for application development.

JumpingBeans from AdAstra [21] tries to address the problem of application development by completely agentifying the applications. The agents are complete applications that jump from machine to machine. While applications can be developed with traditional methods, the migration of complete applications destroys many of the advantages of using mobile agents.

One of the most successful agent platforms is Voyager from ObjectSpace [22]. This platform is an enhanced ORB capable of sending and receiving agents. We believe that the success of Voyager is due to its focus on the application support and not on the mobility issue. This is a move in the right direction. In the ACMAS paradigm we go one step further by providing a very modular and configurable component architecture that supports rapid application development in Visual Development Environments.

The MOA platform from the OpenGroup [23] is a component-based mobile agent platform. While it is based on components allowing different modules to be added and removed, it is still a platform in the traditional sense of the word. The platform is a separate entity from the application. In our approach, we support mobility directly inside the applications.

ADK from TU-Vienna [24] addresses the question of the development of mobile agents by using components. The goal is to simplify the agent development process by using component palettes in Visual Development Environments. The agents are constructed by drag-and-drop of the necessary components from a component palette. While ADK addresses only the agent development, in our approach we are focusing on the application development. In our case, it is the applications and not the agents that are developed using components. This is a very interesting and complementary approach to ours.

5. CONCLUSION AND FUTURE WORK

In this paper we have presented an innovative approach for developing distributed applications based on mobile agents. The ACMAS approach focuses on the support for the application development. Applications are agent-enabled by means of software components that allow them to receive and send agents and provide the necessary services. We believe that this approach will contribute to a faster and easier dissemination of the Mobile Agent Paradigm. The most important reasons that contribute to this are:

- The application can be developed using traditional OO techniques.
- The applications become agent-enable by the simple drag-and-drop of components in a Visual Development Environment.
- The final user does not have to deal with agent platforms but only with applications.
- Integration with ActiveX provides a wide usage platform on where to use mobile agents.

In addition, using a component-based approach brings several benefits when compared to a platform-based approach:

- Agents only migrate to the applications they are interested in, and not to a platform that runs all agents for all the applications. This contributes for having a much more lightweight and robust system.
- By creating components that represent new services, the framework can be easly extended.
- Mobile agents are just like any other middleware and coexists pacifically with other programming tools. This is specially important when supporting legacy applications.

Finnaly, in this paper we have also overviewed the implementation of the mobility component in MOBLITY, and its integration with ActiveX. The support of ActiveX is very important since it brings the mobile agent paradigm into languages and environments currently not supported in other MA frameworks.

We are currently working on higher-level components for providing support for network management. Our intent is to validate the ACMAS approach by building a complete network management application. Building this application will certainly provide many insights on the advantages and difficulties of using a component-based approach for develop mobile agent applications.

ACKNOWLEDGEMENTS

This investigation was partially supported by the Portuguese Research Agency FCT, through the program PRAXIS XXI (scholarship number DB/18353/98) and through CISUC (R&D Unit 326/97).

REFERENCES

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

[2] J. Baumann, F. Hohl, K. Rothermel, and M. Strasser, “Mole - Concepts of a Mobile Agent System,” *World Wide Web*, vol. 1, iss. 4, 1998.

[3] M. Breugst, S. Choy, L. Hagen, M. Hoft, and T. Magedanz, “Grasshopper - An Agent Platform for Mobile Agent-Based Services in Fixed and Mobile Telecommunications Environments,” in *Software Agents for Future Communication Systems*: Springer-Verlag, 1998.

[4] T. Wash, N. Paciorek, and D. Wong, “Security and Reliability in Concordia,” presented at 31st Annual Hawai International Conference on System Sciences, Hawai, 1998.

[5] D. Milojicic, “Mobile Agent Applications,” in *IEEE Concurrency*, vol. 6, iss. 3, 1999, pp. 80-90.

[6] C. Szyperski, *Component Software, Beyond Object-Oriented Programming*: Addison-Wesley, 1998.

[7] "AdventNet SNMP," http://www.adventnet.com/products/snmpbeans, 2000.

[8] "Java Remote Method Invocation - Distributed Computing for Java," Sun Microsystems, 1998.

[9] K. Arnold, B. Osullivan, R. Scheifler, J. Waldo, A. Wollrath, and B. O'Sullivan, *The Jini Specification*: Addison-Wesley, 1999.

[10] R. Orfali, D. Harkey, J. Edwards, and R. Crfali, *Instant CORBA*: John Wiley & Sons, Inc, 1997.

[11] G. P. Picco, "muCode: A Lightweight and Flexible Mobile Code Toolkit," presented at Second International Workshop on Mobile Agentes (MA'98), Stuttgart, Germany, 1998.

[12] "JavaBeans Specification 1.01," Sun Microsystems, 1997.

[13] "The Bean Development Kit": Sun Microsystems http://java.sun.com/beans/software/bdk_download.html, 1998.

[14] D. Rogerson, *Inside COM*: Microsoft Press, 1996.

[15] "Microsoft Windows Products Homepage," http://www.microsoft.com/windows/default.asp.

[16] D. Kruglinski, S. Wingo, and G. Shepherd, *Programming Visual C++*, Fifth ed: Microsoft Press, 1998.

[17] F. Balena, *Programming Visual Basic 6.0*: Microsoft Press, 1999.

[18] S. Teixeira and X. Pacheco, *Delphi 5 Developer's Guide*: Sams, 1999.

[19] "Using JavaBeans with Microsoft ActiveX Components," Sun Microsystems, 1999.

[20] M. Strasser, J. Baumann, and F. Hohl, "Mole - a Java based mobile agent system," presented at 2nd ECOOP Workshop on Mobile Object Systems, Austria, 1996.

[21] "Jumping Beans White Paper," Ad Astra Inc., 1999.

[22] "Voyager ORB 3.2 Developer Guide," ObjectSpace Inc., 1999.

[23] D. Milojicic, D. Chauhan, and W. laForge, "Mobile Objects and Agents (MOA), Design, Implementation and Lessons Learned," presented at 4th USENIX Conference on Object-Oriented Technologies (COOTS), 1998.

[24] T. Gschwind, M. Feridun, and S. Pleisch, "ADK - Building Mobile Agents for Network and Systems Management from Reusable Components," presented at Agents Systems and Applications/Mobile Agents (ASA/MA99), Palm Springs, USA, 1999.