BUILDING DOMAIN-SPECIFIC MOBILE-AGENT PLATFORMS FROM REUSABLE SOFTWARE COMPONENTS

Paulo Jorge Marques*, Luís Moura Silva, João Gabriel Silva

CISUC, University of Coimbra, Portugal
Dep. Eng. Informática – Polo II
Universidade de Coimbra
3000 Coimbra, Portugal

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

Abstract. *Mobile agents are a promising technology for developing applications in many application domains. However, it has not yet gained a large acceptance from the developers. One important reason for this is the difficulty of using generic mobile-agent platforms in specific application domains, which have very concrete requisites. In this paper, we present a reusable component-based framework that enables the creation of domain-specific mobile-agent systems in an easy and rapid way. The framework was implemented using the JavaBeans component model, and a fully blown agent platform was built from the components. By using this framework, the development of domain-specify mobile agent platforms is simplified, allowing a more rapid development cycle, which can contribute to an easier spreading of the mobile agent technology.*

1. INTRODUCTION

Mobile agents (MAs) are a promising new technology for developing distributed applications in several different application domains. Examples include network management, information gathering and filtering, mobile computing, electronic commerce and others [1, 2].

Over the last few years we have observed the proliferation of available generic mobile agents platforms like Aglets [3] from IBM, Voyager [4] from Objectspace, Grasshopper [5] from IKV++, and others. According to the Mobile Agent List [6], a central repository where authors can register the existence of their mobile agent systems, there are now over seventy-two known mobile agent platforms. This is a lot more than available RPC and CORBA ORB [7] implementations. Nevertheless, the technology has not yet achieved mainstream and there are only few known commercial products based on mobile agents.

* 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).

One important question to be asked is: Why there is this proliferation of mobile agent infrastructures, and why there are not more applications available? These questions are especially interesting since most prototype applications work well, and are able to show the advantages of using MAs when compared with other solutions. The most common answers given to these questions are security concerns and the difficulty of developing agent software [2].

We believe that one important reason delaying the deployment of agent applications is that current mobile agent platforms are too monolithic, not adapting well to different application domains. When a developer is faced with the possibility of developing an application that uses mobile agents, the existing generic agent platforms do not provide the set of functionalities needed for his specific application. At the same time, these platforms have a large bundle of functionalities that are not necessary at all.

Another problem regarding the currently available mobile agent platforms has to do with the "You Ignore Related Technology" pitfall [8]. Current MA platforms force the programmer to center all the development on the agents. Agents should be a small part of the distributed applications, not its corner stone. Currently, there is little or no support to develop applications that use MAs only as a small part of its infrastructure. Instead, the applications themselves have to be coded as agents, or interface agents have to be written to stand between the mobile agents and the applications. This can be very restrictive.

We have identified these two important lessons during a previous project called `James` [9]. The objective of the project was to accomplish a MA infrastructure for the management of telecommunication applications, and was developed in collaboration with two industrial partners: Siemens SA and Siemens AG. For creating the infrastructure, we have identified the need to develop a mobile agent platform from scratch since the existing systems did not adapt well to our application domain. We also had to design a remote API that enabled the decoupling of the agent infrastructure from the applications, allowing them to use other distributed technologies besides agents. After having developed the platform, we concluded that it was well adapted for its application domain, but was not very reusable in other domains.

In this paper, we present a component-based framework that allows the overcoming of the limitations previously identified. This framework allows domain-specific platforms to be assembled from reusable binary software components [10]. This assembling can be done using an Integrated Development Environment (IDE) by the simple drag-and-drop of the necessary components, and by configuring their interconnections. This framework was designed to allow a large decoupling between the applications and the platform so that agents are used only on a required-basis. We denominate this component-based approach as the BRICKS approach.

The rest of this paper is organized as follows: Section 2 details the BRICKS approach. Section 3 overviews the implementation of the reusable components.

Section 4 presents the implementation of a fully blown mobile agent platform. Section 5 discusses the related work. Finally, Section 6 presents the conclusions and future work to be done.

2. THE BRICKS APPROACH

In order to quickly build domain-specify mobile agent platforms, an agent framework must have three important characteristics:

- The modules must be highly reusable.
- The framework must be easily extensible.
- The applications and agent platform must interact easily.

Generic MA platforms typically fail on these three items. Normally, the set of features present in a platform are fixed, not allowing modules to be easily added or removed. In addition, the existing modules are normally not reusable for building other systems since the platforms are coded in a monolithic way. Finally, in most MA platforms the applications themselves have to be written as agents, which can be very limiting.

To overcome these limitations we developed the BRICKS approach. In this approach, the modules typically found on a MA platform are implemented as independent binary software components. The systems are assembled in an IDE, where the programmer selects the necessary components from a component palette and configures their properties and interconnections.

2.1 Three different types of components

For developing an agent platform, three sets of distinct components can be identified (Figure 1):

- Mobile-agent support components.
- Third-party off-the-shelf components.
- Domain specific components.

Mobile-agent support components provide the basic needs in terms of mobile-agent infrastructure. There are components for supporting the migration of agents between platforms, components for supporting different inter-agent communication mechanisms, components for agent tracking and so on.

Third-party off-the-shelf components are components commercially available from software makers that can be used for building the system. Currently there is a large variety of components available for the most different things, like accessing databases, designing graphical user interfaces, messaging and others. All these components can be used for building the platform without having to re-implement the required functionalities.

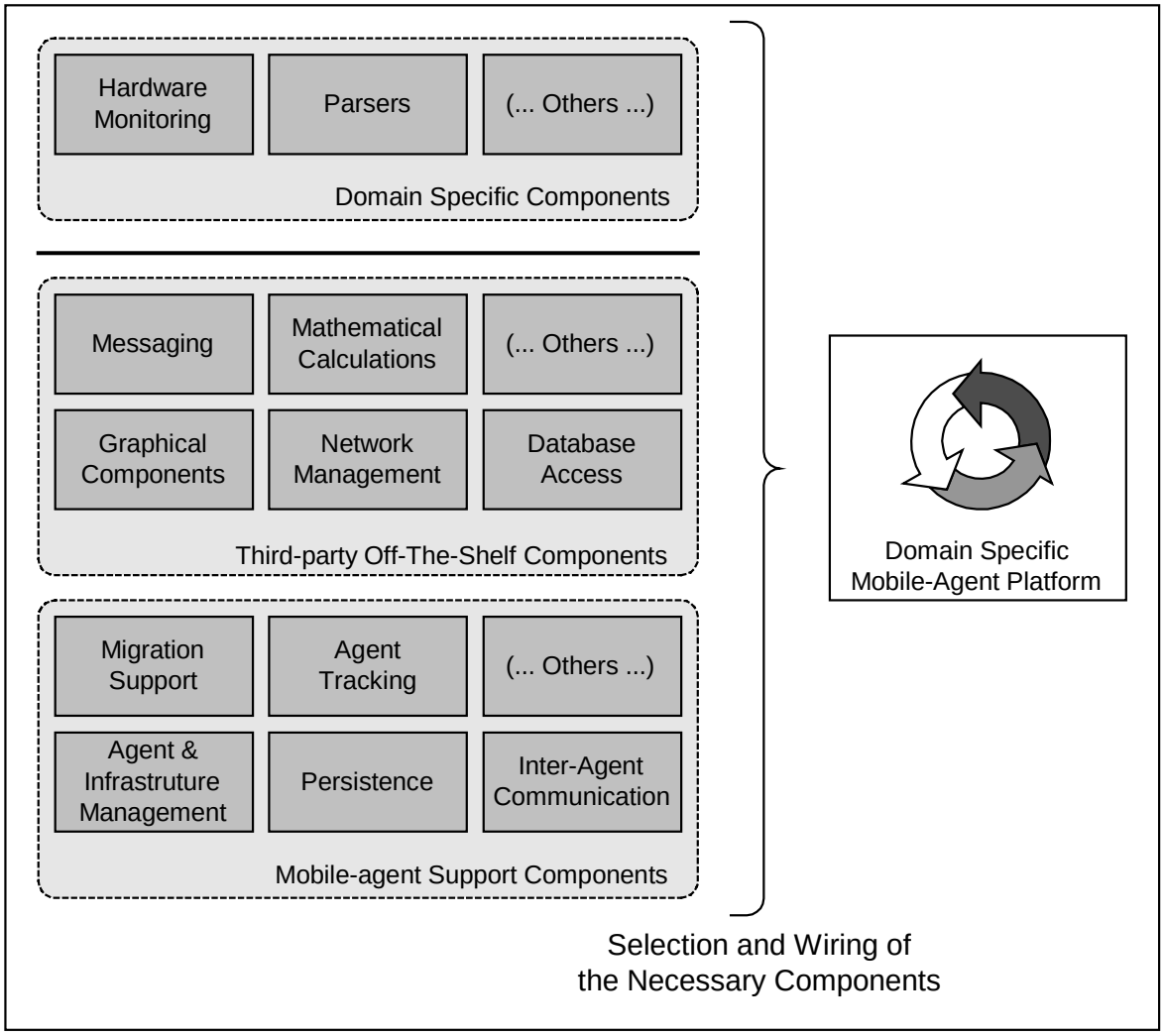

Figure 1 – Assembling a mobile-agent platform from reusable components.

Domain specific components are modules that must be written in the context of the application domain being considered, providing functionalities not readily available off-the-shelf. For instance, it may be necessary to write special parsers for extracting information from files, or to write supporting services for allowing agents to monitor the hardware of a machine. These modules can be coded as components and incorporated into the platform.

It is important to distinguish two types of components: (a) components that provide new services at the platform level, and (b) components that provide new services for the mobile agents. An example of the first is a component that provides agent-tracking services, allowing the agent platform to know where an agent is at any moment. An example of the second is a component that provides a new inter-agent communication mechanism, allowing the agents to exchange messages in a different way. One of the major reasons why generic mobile-agent platforms cannot be reused in many application domains is that they are not

extensible. One important objective of using a component-based approach is to be able to incorporate new services as components in an easy and flexible way.

3. IMPLEMENTING THE COMPONENTS

We will now overview how the BRICKS approach was implemented using the JavaBeans component model [12]. We will concentrate on the Mobility Component that allows agents to migrate between platforms and provides the means for easy extensibility.

The Mobility Component is coupled with the platform application and with the other components through events (Figure 2). There are two major sets of events: *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 platform). Service Support Events enable the Mobility Component to incorporate new services and to be notified if they are no longer available.

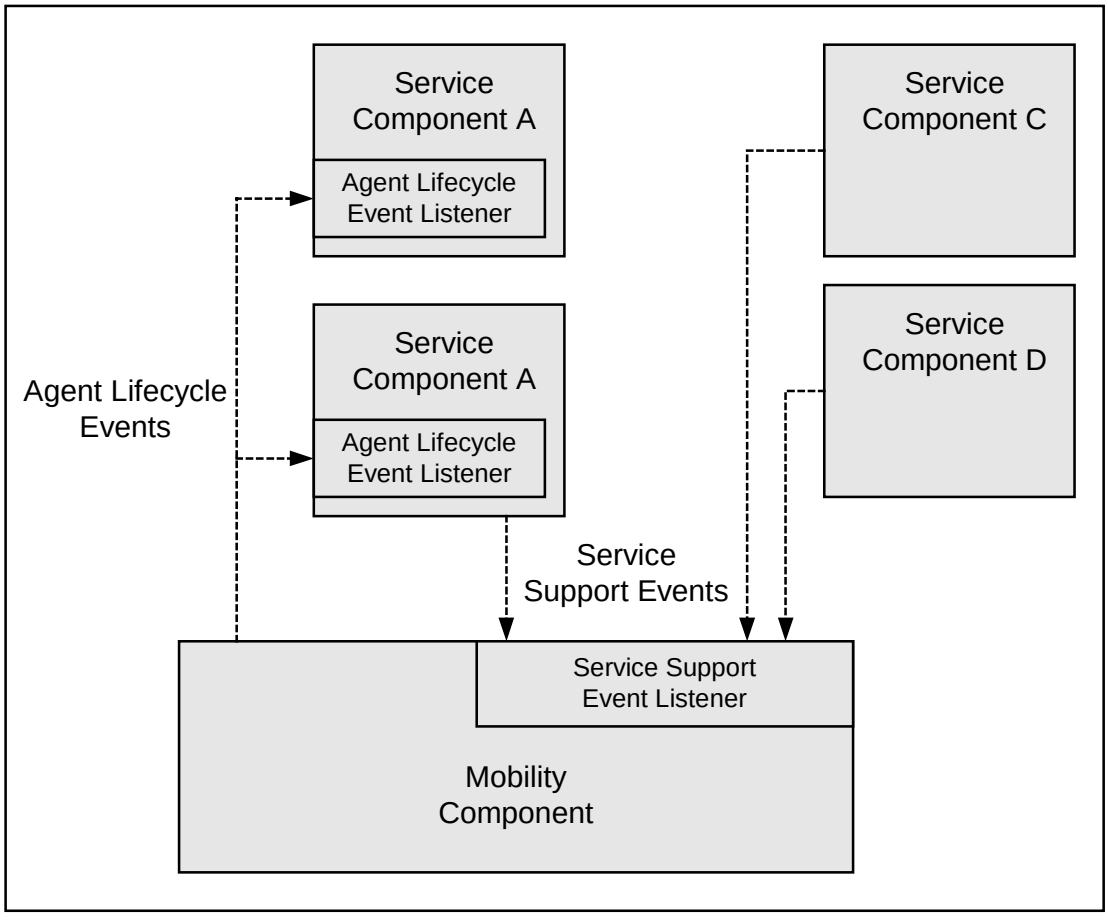

Figure 2 – Interactions between the Mobility Component and Service Components.

Whenever an agent changes its execution state (e.g. arrives, departures or dies), an Agent Lifecycle Event is fired. The platform 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 platform. This is especially important for the higher-level services since they can make use of those events to accomplish their functions. For instance, a disk-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 allow a service to be incorporated or removed from the system, and enables the services to be configurable in an IDE. 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. If the service needs to be removed from the list of available services, it fires an event requesting to be removed from the list of available services. The existence of this event set is extremely import because if allows development tools like the Beans Development Kit (BDK) [11] to setup adapters between components.

All the features of the Mobility Component are accessible through properties [10]. Thus, for instance, it is possible to configure incoming and outgoing TCP/IP ports, 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 an IDE. Figure 3 shows the mobility component being used on a visual development environment.

We have found that these two sets of events are enough to guarantee the requirements of having an agent framework that easily adapts to several application domains:

- Each component is highly reusable, only being incorporated into the platform if needed.
- It is easy to develop new components that offer agents new services, being aware of the activities developed by the agents.

In order to allow the applications to easy interact with the agent platform, we have also created a component that allows an application to remotely manage the infrastructure through the mobility component (e.g. change the component listen ports or shutdown it). Besides allowing the control of the infrastructure, this component also enables an application to perform actions on the agents, like launching them, control their execution state and termination.

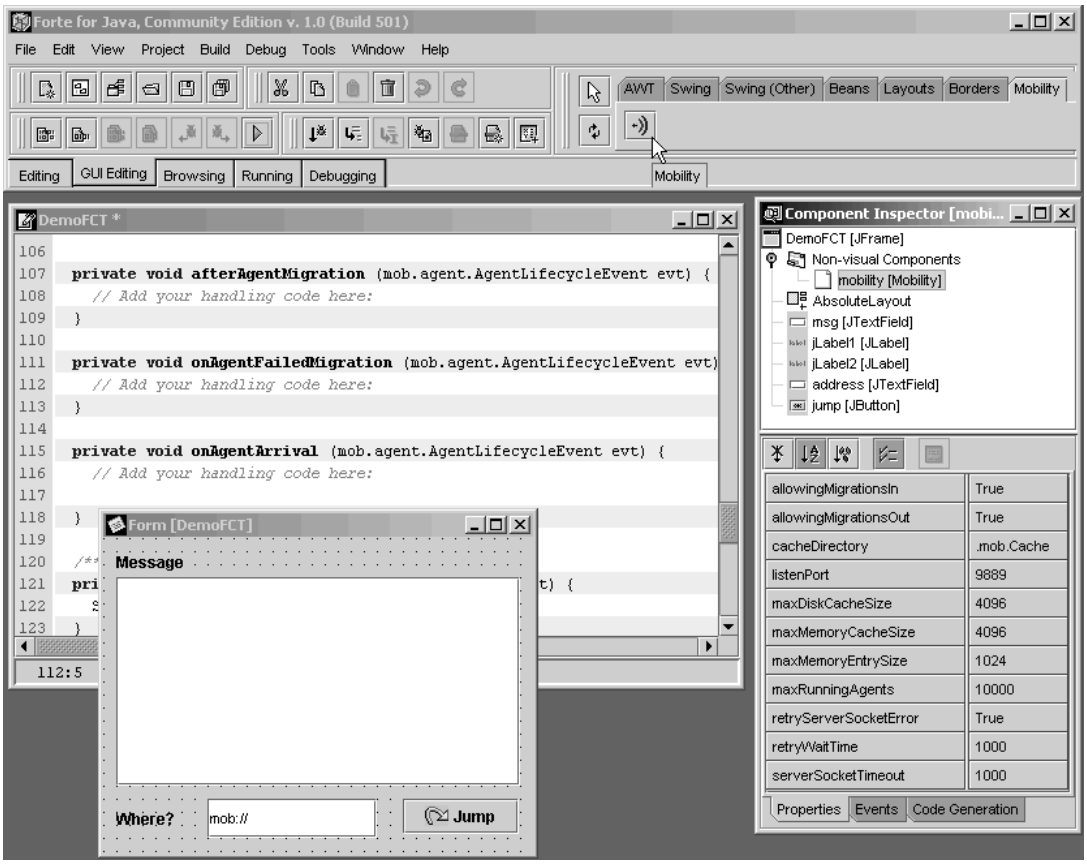

Figure 3 – The Mobility Component being used in an IDE.

We currently have a rich component palette that covers many of the functionalities typically found on traditional MA platforms (e.g. components for several inter-agent communication paradigms, components for managing the infrastructure, components for agent tracking, among others).

4. IMPLEMENTING AN AGENT PLATFORM

In order to test the feasibility of rapidly building an agent platform from components, we have implemented one. This platform also gave us a useful test bed for debugging the available components. Figure 4 shows the Graphical User Interface that controls the platform.

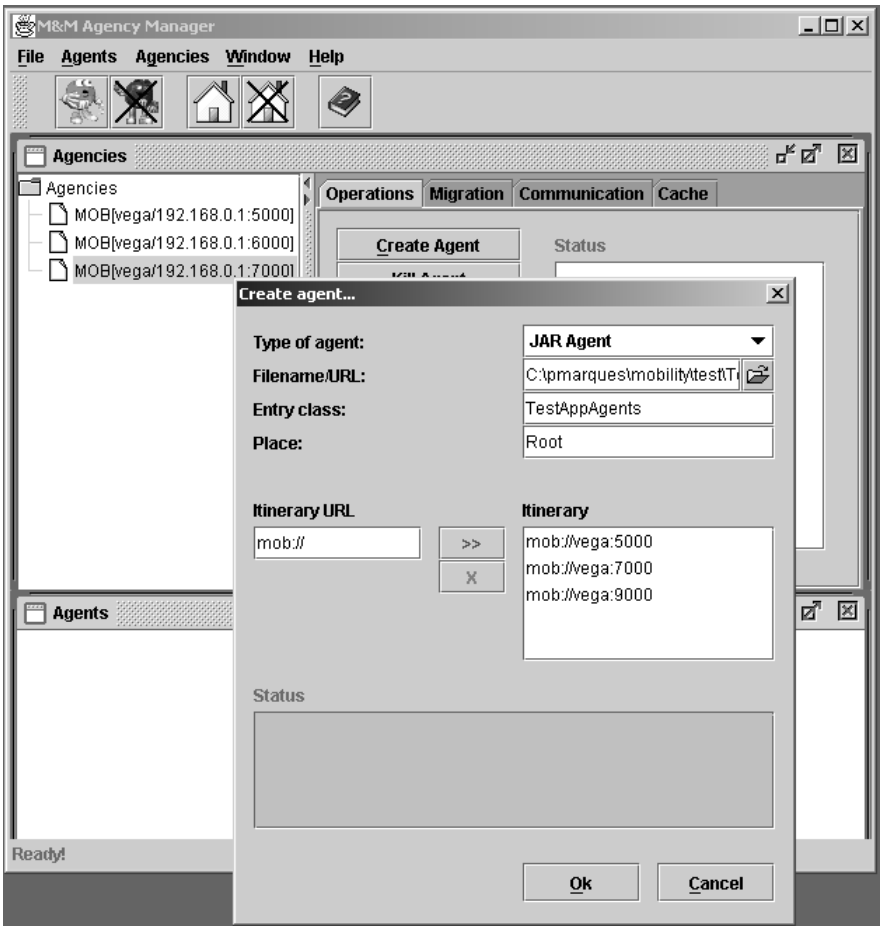

Figure 4 – A generic MA platform built from reusable components.

This platform was developed by simply connecting and configuring the existing components. The only major difficulty that we have found was the lack of means to run it as a daemon and remotely control it. To address these problems, we have developed two different modules:

- The Agency.
- The Agency Daemon.

The Agency is the base container where the several components are included. This container provides a remotely available interface that allows the applications to control the Agency. The remote interface is implemented using the management component mentioned on Section 3, and by setting up a reference in a common registry so that remote applications can easily locate it. This common registry in managed by the Agency Daemon.

The Agency Daemon is an NT service that performs two tasks:

- Sets up a registry that contains references to the management interface of each Agency present in the system.
- Allows Agencies to be remotely started, shutdown and queried.

Figure 5 depicts the approach. An application can access the registry in the Agency Daemon and ask for an agency to be started up or shutdown. Additionally, it can query the daemon for the Remote Management Interface of an agency. Once it has this interface, it can launch agents, stop agents, control the running services and all the parameters of the agency.

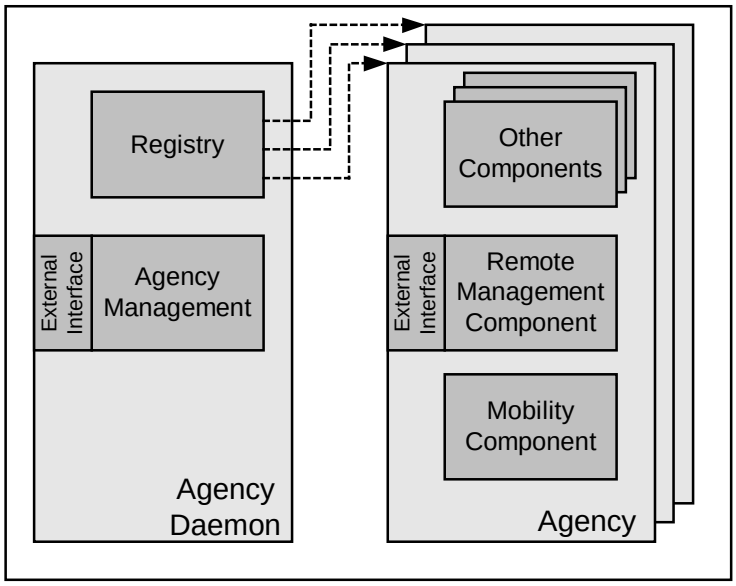

Figure 5 – The Agency Daemon and the Agencies.

The Agency and the Agency Daemon modules are generic and can be reused for building new domain-specific mobile agent platforms.

5. RELATED WORK

One of the most successful agent platforms is Voyager from ObjectSpace [4]. This platform is an enhanced ORB capable of sending and receiving agents. Voyager is not extensible from the point of view of the programmer, since its features are fixed and there are no readily available means for extensibility. Nevertheless, since it is an ORB, some degree of extensibility could be obtained by implementing some services at the application level. Even so, these services would be quite restrict since there is no straightforward way of providing new services for the agents themselves.

The MOA platform from the OpenGroup [13] is a component-based mobile agent platform. While it is based on components allowing different modules to be configured, MOA is still a full-flagged mobile agent platform. Since MOA was an industrial project, there is not much available information concerning its extensibility and its component-based aspects. Even so, one fundamental difference between our work and the MOA project is that we specifically address the problem of system extensibility. We do not provide an agent platform, but a

palette from where agent platforms can be assembled. From what we could infer, MOA is a configurable agent-platform.

ADK from TU-Vienna [14] 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 an IDE. The agents are built 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 platform development.

6. CONCLUSION

In this paper, we have presented a binary software component framework that allows the construction of domain-specific mobile agent platforms. The framework was implemented using the JavaBeans technology and achieves three key points that make the construction of domain-specific platforms easier:

- The existing components are self-contained and reusable.
- The framework is easily extensible and configurable.
- The applications and the agent platform interact easily.

In order to test the approach, we have developed a mobile agent platform that embodied the existing components and additionally integrated a GUI for its management. This turned out to be a simple task, which make us believe that the approach is generally applicable.

We are currently addressing the problem of making the deployment of components more flexible, making them available at the network level and being dynamically deployed at run-time. JINI is being considered as a possibility for implementing these features.

REFERENCES

[1] D. Lange and M. Oshima, *Seven good reasons for using mobile agents*, Communications of the ACM, vol. 42, issue 3, 1999, pp. 88-89.

[2] D. Milojicic, *Mobile Agent Applications*, IEEE Concurrency, vol. 6, issue 3, 1999, pp. 80-90.

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

[4] ObjectSpace Inc., Voyager ORB 3.2 Developer Guide, ObjectSpace Inc., http://www.objectspace.com, 1999.

[5] 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*, Software Agents for Future Communication Systems, Springer-Verlag, 1998.

[6] *Mobile Agent List*, http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html, 2000.

[7] R. Orfalli, D. Harkey, and J. Edwards, *Client/Server Survival Guide*, 3rd, John Wiley & Sons, Inc, 1999.

[8] M. Wooldridge and N. Jennings, *Software Engineering with Agents: Pitfalls and Pratfalls*, *IEEE Internet Computing*, vol. 3, issue 3, 1999.

[9] L. M. Silva, P. Simões, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, and N. Stohr, *James: A Platform of Mobile Agents for the Management of Telecommunication Networks*, 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99), Stockholm, Sweden, 1999.

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

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

[12] Sun Microsystems, *JavaBeans Specification 1.01*, Sun Microsystems, http://java.sun.com/beans/docs/spec.html, 1997.

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

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