Mobile Agents Platforms over OSGi

Mikael Desertot, Si-Hoàng Do, Didier Donsez, Marc Bui

Abstract—**This paper presents two approaches for developing mobile agent architecture on OSGi gateway: one relying on the IBM standard (Aglet) and the other using only the standard OSGi design. Both use OSGi dynamic capabilities for dynamic code loading and unloading and take advantage of service oriented architecture for binding mobile agent requirements with services already running on the hosts. Those approaches show the interest of services architecture to help and facilitate mobile agent infrastructure implementation.**

Index Terms—**service oriented architecture, OSGi, mobile agent platform.**

I. Introduction

NOWADAYS, we assist at the emergence of new heterogonous distributed systems. For instance, appearance and development of residential gateways that can be accessed by the PDA or the mobile phone of the proprietary is one of those networks. This drives to a growing interest for the services platforms that can be deployed in such environment and can be used either on the client side (PDA, mobile phone), either on the server side (gateway). OSGi[1][23] is one of those service platforms. It can fit the memory requirements of those embedded devices because of its small footprint.

It is still hard to take advantage of such environment because of the difficulty to develop and implement distributed systems whereas the demand for added value services is growing. This demand can be considered as a new opportunity for mobile agent platforms to democratize. We claim that we can rely on OSGi service platform (gateway) for pervasive computing (mobile phone, interactive television, in-vehicles…) and provide a mobile agent environment.

A mobile agent is a composition of computer software and data which is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer. By moving computation to data, it can reduce the network load. This computation paradigm can also provide dynamic adaptation (actions are dependant from the host environment), network fault tolerant (can be efficient without a permanent active connection between the server and the client) and flexible (only the agent source should evolve, not the hosts servers).

The idea that technology is moving beyond the personal computer to everyday devices with embedded technology and connectivity as computing devices become progressively smaller and more powerful. Also called ubiquitous computing, pervasive computing is the result of computer technology advancing at exponential speeds - a trend toward all man-made and some natural products having hardware and software. Pervasive computing goes beyond the realm of personal computers: it is the idea that almost any device, from clothing to tools to appliances to cars to homes to the human body to your coffee mug, can be imbedded with chips to connect the device to an infinite network of other devices.

…

II. Mobile Agents Platforms over OSGi

The Open Services Gateway Initiative (OSGi) specification is an open framework for the delivery of managed broadband services to networks in homes, cars, mobile phones, factories and other non-stop environments. Services are packaged inside deployment units named "bundles," where a bundle is a collection of Java classes and associated resources. Abstractly, the OSGi specification is divided into two logical halves: the OSGi framework and OSGi services. The framework defines the lifecycle of the bundles and the registration of services. The services are either general-purpose services using by main applications developers such as HTTP, Log, Device Manager, JINI or domain-oriented services such as WireAdmin for sensor-based applications.

A. Aglets over OSGi

The OSGi platform has been applied in many domains such as vehicles, mobile/portable devices, offices and homes. There are a lot of services have been developed. However, the OSGi is a non-distributed service platform. Then, our purpose is to build an over-platform that permits the cooperation between OSGi gateways and supplies an ability of creating a new type service that is more flexible and intelligent. A mobile agent framework is the most suitable for this over-platform.

Building the mobile agent framework relies on OSGi platform gives mobile agents in different OSGi gateways and in normal hosts communicating (locally and distantly) and migrating from one host to another as Fig 2.1.

For the implementation such mobile agent framework, we have chosen Aglets platform of IBM [10] that is a strong, multi-applied, Java and open source framework.

All of communication, migration and security mechanisms of mobile agent framework over OSGi [2] inherit from Aglets framework. We will talk in details below.

Published to : Mikael Desertot, Si-Hoang Do, Didier Donsez, Marc Bui , "Mobile Agents Platforms over OSGi", Proc. of 4th International Conference on Computer Sciences, Research Innovation and Vision for the Futur (RIVF'06) February , 12-16, 2006, Ho Chi Minh, Vietnam

1) *Inter-agent communication*

Agents (aglets) communicate by sending and receiving messages. For sending message agent has to know the receiver's ID and can be returned a reply value. There are three modes of sending messages: One-way, synchronous and asynchronous mode.

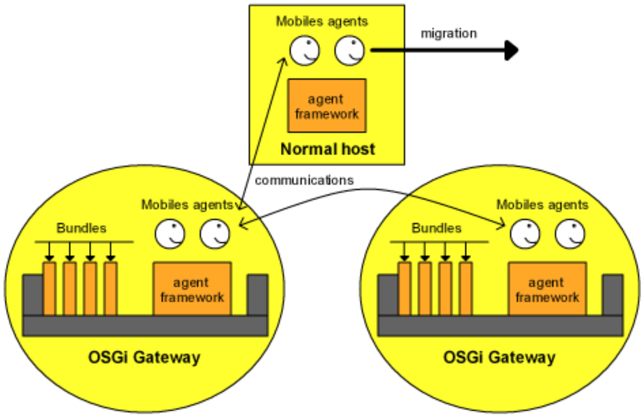

Fig. 2.1. Mobile agent framework over OSGi.

Aglets API supplies three methods for sender agent that are:

- *proxy.sendOneWayMessage(msg);*
- *proxy.sendMessage(msg);*
- *proxy.sendAsyncMessage(msg).*

By receiving and treating incoming messages, every agent has the *handleMessage(msg)* method. All messages are first passed to the *MessageManager* of the mobile agent framework. Here, they are queued for later processing in order.

The message's content is organized in a hash table of couples of value (*name, value*).

2) *Migration*

The aglets' migration is realized through Agent Transfer Protocol (ATP). Every mobile agent framework is an ATP server. There are two main activities for the agent's migration: *Dispatch* and *Retract*.

Dispatch requests a destination agent framework to reconstruct an agent from the content of request and to start executing agent. If the request is successful, the sender must terminate the agent and release any resources consumed by it.

Retract requests that a destination agent framework send the specified agent back to the sender. The receiver is responsible for reconstructing and resuming the agent. If the agent is successfully transferred, the receiver must terminate and release any resources consumed by it.

Intuitively we just need the *Dispatch* activity for agent's migration. However, in case of firewall, only one-way connection is established to the outside. Agent can not be returned relies on the *Dispatch* activity. And then the *Retract* activity has done.

3) *Security*

The security model is based on the Java 2 security specification. It defines security policies to agent owner, context owner, and network domain owner.

The main objective of agent owner is to protect the agents from attacks. When agents are visiting a context they may request the context to enforce the specific policy. The context owner is above the agent owner in the security hierarchy and need not wholly agree with the agent owner's security policy.

A context authority is responsible for keeping the server and the underlying system safe from malicious agents. The context owner's security policy defines the actions an agent can take in a given context.

A network domain authority is responsible for keeping its network secure so that the server within the domain can provide their services safely and all incoming agents can complete their tasks. The domain authority defines the security policy for the domain.

B. *Mobilet[24] over OSGi*

A different approach from the Aglet, by just relying on the OSGi standard, is to consider that a mobile agent is a bundle that encapsulates the agent business code. In this manner, due to the needs of dynamic and distant code loading, we profit of all mechanism integrated in the OSGi platform to implement such a mobile agent environment and benefit of the standards services provided by the specification.

The capabilities we are aiming to reach are describe in the fig. 2.2. An OSGi gateway is assumed to be installed on each server that we want our mobile agent to attain. On each one, a bundle is deployed (a bootstrap) and will help agents to progress through the whole of the servers potentially being able to lodge them. Currently, to implement such a bootstrap, a servlet responds to our needs.

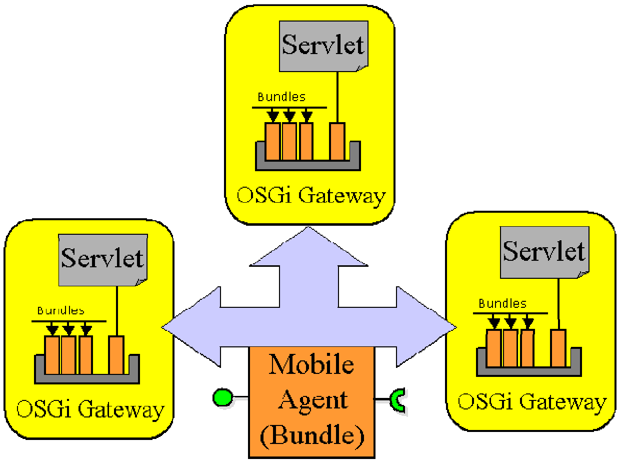

Fig. 2.2: Deploying mobile agent on OSGi

We have defined interfaces that both server and client (mobile agent) must implement. Those server side classes are obviously contained in one of the server bundle deployed on the OSGi platform but another bundle should also contain the client interfaces needed by the mobile agent. This avoids embarking the client classes in the moving bundle and this for two reasons. First, for a performance purpose, as less classes have to move from one platform to another and second, for a security purpose, as class duplication in different bundles is not suitable and won't work in OSGi situation. This approach is less interesting in case the client API evolves frequently as bundles of alls platforms have to be updated whereas if the client brings them, this action is simplified. Packaging them separately is interested for independent updates.

To start a mobile agent, we connect to any of our OSGi bootstrap, using a web browser. Then we enter the URL of the mobile agent location. This agent is package as a bundle and will be started by the server we are connected to. OSGi loading mechanism takes in charge downloading the bundle and installing it. As soon as its package dependencies are resolved (interfaces exported by the bundle server), the mobile agent is started. The bundle Activator can then fulfils several functions. At first it must instantiate a tracker on a log service that should be present on the platform. If this service is not available the agent won't be able to log information but will keep on working. The work done, the agent must connect to the send agent service proposed by the server. The agent Activator try to load the files containing his properties and, if it exists, the file containing the serialized information the agent brings during his transfer. If this last file doesn't exist, it's the agent first activation. Finally, the agent is initialized. This part implies in particular during the first installation, the setting of the properties and the read of the different jumps that must be carried out including their name resolution.

Through this architecture, we are now able to propose different scenarios: migration, meeting, waiting, waking up and a server agent specific scenario of auto identification.

During its execution, the client asks for migration. If the jumps are explicitly established in the file properties, the agent will use this definition and will manage a table containing the jumps and their state (carried out or not). If the jumps are not specified in the properties file of an agent, the jumps will be made on all running Agent Server. The home host, if specified, will be the last to receive the agent. The trace of the jumps is preserved. This is the classical migration scenario.

In a meeting scenario, agent looks for a named specific agent locally. If it can't locate it ask other servers to list agent installed on their location. Askers will then use their responds to move to a server where the requested agent is accessible.

For a waiting scenario, an agent requires the presence of another agent to perform his operations. So it is possible for it to sleep until such an agent registers it on the same server. When conditions are met begins a waking up scenario to defreeze the sleeping agent which is know able to run properly with the help of the freshly arrived agent.

For agent server purpose is provided an auto identification scenario permitting to any new server to be registered by server ever running. We use for this scenario a multicast diffusion. A new running server diffuses his name and identification on the network so all other servers take in consideration this arriving.

We have implemented such a platform using the OSCAR [24] open source implementation of OSGi. It is available at www …

1) Code and state Packaging

Two specificities should be pointed there; resolving the agent code package dependencies and the way the state of a mobile agent is saved and migrated within the agent bundle.

A mobile agent, depending on the actions it performs, would require different packages to work properly. We can not assume what is already deployed on a gateway but we can take advantage of OBR (Oscar Bundle Repository) [25]. It has two goals; first, it simplifies the deployment and the use of existing OSGi bundles. This objective is achieved by the providing of a service that can automatically install a bundle, with its deployment dependencies, from a bundle repository. Second it encourages independent bundle development by raising the visibility of the available bundles and providing access to both the executable bundle and its source code. By using this service we are able to resolve agent package dependencies and automate their installation.

The state of a mobile agent must be saved and reloaded each time the agent migrates. For this part we are serializing needed information and generating a bundle containing them which become the new mobile agent that will be installed by the next jump platform.

2) Inter-agent Communication

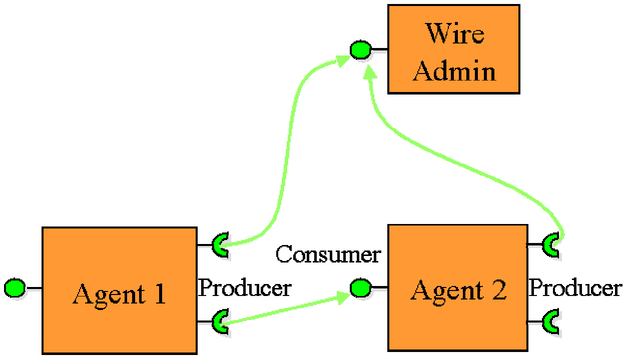

Fig. 2.3: Connection through Wire Admin

The OSGi specification since its third release, propose an architecture called "wire admin" that permits to connect an information producer with a consumer requiring this kind of information. Wire Admin (figure X) is a service proposed to every bundle being installed on OSGi platform. It can be used to create Wire objects connecting a Producer service and a Consumer service. Wire objects also have wire properties that may be specified when a Wire object is created. The Producer and Consumer services may use the Wire object's properties to manage or control their interaction.

As our mobile agent travel on OSGi gateways, we can have benefits to use this standard architecture for inter-agent communication. Indeed, in the case of the meeting scenario evoked upper, agents can use the producer/consumer pattern to communicate and share information. Each agent would be both a producer and a consumer. The characteristics of the services they need or proposed would be typed as flavors for the Wire Admin so that it is able to connect two agents advisedly. With the use of Wire Admin we use the facilities proposed by the OSGi platform at its best but one more problem subsist in this case: as our agent are self dependants, they won’t connect automatically to the Wire Admin at their arrival on a gateway because this operation required an administrator command to be performed, whereas in our case this is not realizable. What we propose is to automate the connection by introducing a Wire Admin Binder. This service, on top of Wire Admin, take the role of an administrator and uses the Wire Admin capabilities to instantiate a Wire Object between two mobile agents, getting sure as a preliminary that the flavors associated with data properly fit.

3) Migration

Agent migration is realized through HTTP. Each agent server registers a servlet. An agent uses it to ask his destination server for migration.

This demand is performed after the agent has been repackaged into a bundle. This bundle is exposed by HTTP. Concretely it is the URL of this bundle that is sent to the destination server. It can then rely completely on OSGi mechanism for code downloading. And again the bundle's Activator class gets the needed services on the server to deploy and carry out its operations. This operation is repeated as much as all the jumps have not been traveled through.

4) Security

Security handling has been one of the major improvements of the fourth release of the OSGi specifications. It provides a rich set of capabilities on which we can rely to obtain the security level we want.

A new standard OSGi service has appeared, the Conditional Permission Admin. It provides a simple way to manage networked services securely. It also supports dynamic policies that can depend on external (custom) conditions. It can be combined with the new support for digital signatures. This provides a one stop security solution to agent deployment using the OSGi Service Platform. The agent can embed his proper digital signature that can be checked after a migration and its arrival on a new server to ensure we are not running malicious code.

C. Comparison

The two approaches we have described above don’t provide the same capabilities even if they rely on the same services platform. We have compared both of them using the criteria presented in the table 2.4.

Each of them (Aglets and Mobilet) relies on the OSGi framework and is so able to use Standard OSGi services (described in the specifications) as well as some other services that are registered on the OSGi framework. But as Aglets depends on a specific framework for agent execution, they can’t record a service (defined by a contract) in the OSGi registry that can then be bind and use by a third party bundle. This limitation is broken on the Mobilet architecture, as agents are pure OSGi bundles deploy on the framework in the same way classical bundles are.

Even so, this need of a particular framework for aglets brings interesting capabilities like being able to run on the same context (OSGi gateway) multiple version of the same Mobile Agent. Indeed, the aglet agent framework authorizes the registration of multiple instance of agent but in Mobilet case, as agents are pure bundle, we are not able to deploy two of them on the same gateways. In fact it could be possible if the same bundle can be downloaded from two different places as it is in this way bundles are differentiated in OSGi. In this case, they must also have two different names because the name is used as part of an identifier when agents are repackaged into bundles before a migration. This can lead to conflicts problems.

Criteria	Aglets	Mobilet
Uses OSGi Standard Services	+	+
Can provides OSGi Services	-	+
Collocated execution of the same agent version	+	- (But possible)
Specific politics	+	+
Reusability	+	-
Standard conformance	OSGi Aglets	OSGi
Resource cost	+/-	+

Fig. 2.4: Aglets and Mobilet comparison

Being conform to Aglet also allows a better reusability of our agent, another Aglet compliant framework. By only relying on OSGi inside Mobilet, we are fully dependant of this platform and agent must be adapted to be usable elsewhere.

But using the Aglet framework means that is must be deployed on each OSGi gateway. This cost resources (less than 800ko) and this can be avoided in the Mobilet environment as just the registration of a servlet and the needed is necessary. The underlying OSGi framework already provides all other needed mechanisms (download, installation, activation, services dependency…).

To summarize, relying on two standards (Aglet and OSGi) rather than one (OSGi) (criteria) brings additional functionalities and a better reusability whereas, on the other hand, relying only on OSGi costs less resources and provide a better integration on the underlying platform. The use of one or the other solution depends on the domain specific requirements.

III. Domain experimentations

We apply the Aglets framework over OSGi in domain of comfort services in the SCAM[1] project [3].

The purpose of SCAM is to increase the capacities of the services oriented users, in particular for the nomadic users, and to equip the rooms and buildings with computer functionalities facilitating the comfort management [8][9].

[1] SCAM : Système de Confort utilisant la technologie Agents Mobiles

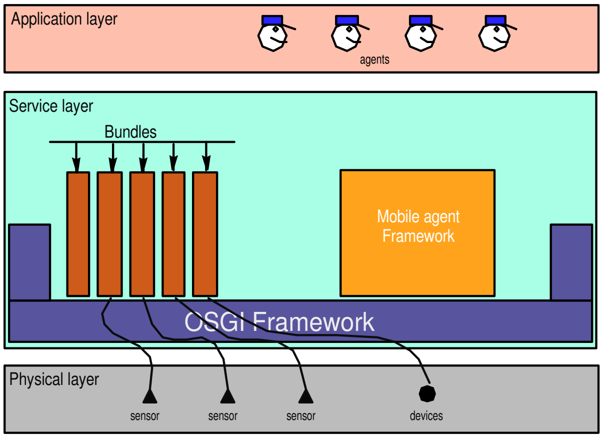

Fig. 3.1: SCAM architecture

SCAM's architecture (fig. 3.1) contains separate physical, service, and application layers.

Physical layer consists of various devices and sensors such as lamps, TV, convectors, motion detectors, heating thermostats.

The service layer contains the OSGi framework and its activated services including three bundles with three different tasks: configuration service, tracking service, and mobile agent framework.

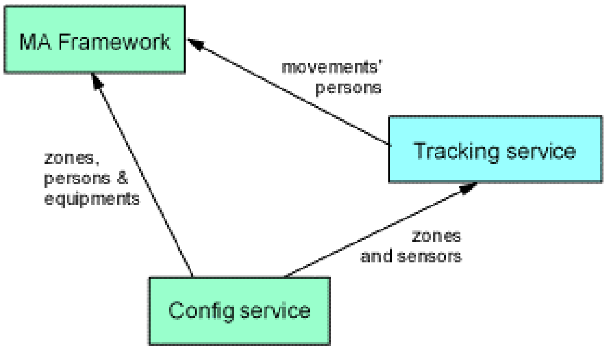

Fig. 3.2: interaction between services

- The configuration service covers the house's configuration to other services. It supplies the information of house's zones and sensors for the tracking service and the information of house's zones, persons in this house (preference, priority …), and equipments for the mobile agent framework.
- The tracking service controls realizing objective sensors. It offers persons' movements to the mobile agent framework.
- The mobile agent framework gets house's information and creates contexts (logic spaces) corresponding to house's zones, initiates assistant agents for each person. When the framework receives the person's movement event, it will dispatch his assistant agents to the corresponding context.

Our system models a logic space correspond to the user house. Each zone in the house equals a context in mobile agent framework. Mobile agent embodies a service for a person. When the person enters a room, his assistant agent will be sent to the corresponding context in the mobile agent framework. At that context, this assistant agent encounters and interacts with other agents.

The system provides a mechanism that follows to develop services where assistant agents can coordinate to reach an acceptable comfort level.

The assistant agent can control the equipments through the existing services in OSGi gateway.

IV. Relate works

This works deals with using a service oriented[21] platform, OSGi, and its deployment capabilities to support a mobile agent environment.

The service oriented architecture is not proper to OSGi and can be found in other architecture. For instance, one of the most classical one is Web Services [13] that rely on distant invocation and a centralized directory. UPnP for its parts retake the same capabilities but with a distributed directory. Those two are service architecture but are not service platform on which we can rely for agent deployment and migration.

JINI[15][16] or OpenWings[17] can be good candidates, more corresponding to service oriented platform because of the notification of services arrival or departure.

JINI seeks to simplify the connection and sharing of devices on a network. It announces itself to the network, providing some details about its capabilities, and immediately become accessible to other networked devices. Under this technology it is possible to create distributed computing, whereby capabilities are shared among the machines on a common network. It can also allows users to access the power and features of any device on the network and would free the desktop computer from holding all the memory, storage and processing power it needs for any job. It has already been used in collaboration with UPnP architecture [14].

Derived from JINI to offer more complex service lookup semantics model is OpenWings. Its goal is to provide a service-oriented component framework for highly dynamic networked systems of software and hardware components. It is based on only one core technology, Java, and is independent from all others. In particular because of it's providing of plug-ins abstractions of these technologies. Component services encapsulate the details of providing and using services.

But those two service platform does not propose the deployment properties OSGi provides by the dynamic deployment of services and their standard packaging into bundles.

Apart those service oriented platform is a project called JXTA [18][19]. This technology is a set of open, generalized peer-to-peer protocols that allows any connected device (from cell phone to PDA, PC or even server) on the network to communicate and collaborate.

Java mobile agents have already been applied to the JXTA P2P platform in this project [12]. It shows that development requirements are minimal by using the platform API and the

Core specifications. It helps program, install and launch mobile agents that are both light-weight and disposable. Such a platform gives the agent the ability to adapt to the unpredictable behavior of the peers and to their content and this content's potential mobility (if either the peer or the content move, both can still be found). Agent deployed on this platform is not encumbered, but rather, adapts its itinerary to the variable network behavior it might encounter. Finally a JXTA P2P agent system doesn't need to be concerned anymore with the underlying real network impediments to communication such as Network Address Translation (NAT) and firewalls.

The JXTA can be compared to the agent framework we propose but again, deployment in OSGi is eased as tackled in the specification, and as it is an open service oriented platform it is possible to use legacy services in our agent to provide enhanced capabilities.

For high-tech residence, Majitek [20] proposed a commercial framework based on JINI technology, called Maji framework. Maji is a platform allowing the development and the management of distributed applications. It enables any device, system, software or service to be virtualized as a common component building block (Meem). Meems are managed by the Maji framework as nomadic resources which can transit on the Maji network (called cloud system) and can be linked together to form complex applications or services that interoperate with other Meems, representing physical devices or systems.

There is the very strong similarity in the management of the Meems' life cycle with the management of the OSGi bundles. However, Maji's graphic resources are mixed with the other components: no separation in application layer. It costs in maintenance phase.

V. Conclusion and perspectives

This paper has presented two approaches for developing mobile agent architecture on top of OSGi gateway, one relying on the IBM standard (Aglet) and another using only the standard OSGi design. The first approach has been experimented in a real use case. Both are using OSGi dynamic capabilities for dynamic code loading and unloading. They also take advantage of the service oriented architecture for binding mobile agent requirements with non functional services already running on the hosts.

Those approaches show the interest of services architecture to help and facilitate mobile agent infrastructure implementation. Thanks to their capabilities to fit embedded environments and the current development of residential network and domotic, we would bet this approach can become a major response of consumers needs.

References

[1] OSGi service gateway specification, release 4, October 2005, http://www.osgi.org

[2] Si-Hoàng.Do, Marc Bui, and Paul Kopff, "A based-on agent framework for Home Networking", RIVF '03, pp.165-168, 2003.

[3] Si-Hoàng Do, Thuy-Liên Pham, "The comfort system agent-based framework and smart lighting controlling application", RIVF '04, pp.165-168 2004

[4] Home Plug and Play: CAL-based interoperability for Home Systems, *CEBus Industry Council*, 4405 Massachusetts Avenue, Indianapolis, IN 46218, USA. 1997.

[5] T.A. Horan, "The Paradox of Place", *Communications of the ACM*, (44) 3, pp.59-60, 2001.

[6] Borriello, R. Want, "Embedded Computation Meets the WWW", *Communications of the ACM*, (43) 5, pp.59-66, 2000.

[7] G. Bell and J. Gemmel, "A call for the Home Media Network", *Communications of the ACM*, (45) 7, pp.71-75, 2002.

[8] Ichiro Satoh, "Bridging Physical and Virtual Worlds with Mobile Agents", *Journal of Information Processing Society of Japan*, vol. 44, no. 8, pp. 2218-2229, August 2003

[9] Kindberg, T., Barton, J., et al., "People, Places, Things: Web Presence for the Real World", *Proceedings of 9th International World Wide Web Conference (Www9)*. 2000.

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

[11] D.B. Lange, and M. Oshima, "Seven good reasons for mobile agents", *Communications of ACM*, 42(3). pp. 88-89, March 1999.

[12] Rita Yu Chen and Bill Yeager, "Java Mobile Agents on Project JXTA Peer-to-Peer Platform", *Proceedings of the 36th Hawaii International Conference on System Sciences* (HICSS'03), IEEE, 2002.

[13] Klaus-Peter Eckert, "The Fundamentals of Web Services", *The Industrial Information Technology Handbook*, 2005

[14] J. Allard, V. Chinta, S. Gundala and Golden G. Richard, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability", *Symposium on Applications and the Internet (SAINT),* 2003

[15] Hugo José P. B. Paulino Pinto, "Distributed Networking Computer Model: SUNs JINI as an Advance in the Technology-An Overview of Javas Distributed Platform", *ICEIS,* 1999

[16] Jim Waldo, "JINI Networking Technology and Ad-Hoc Networks", *15th Conference on Systems Administration (LISA)*, December 2001

[17] OpenWings, Web Site, http://www.openwings.org/

[18] Li Gong, "Industry Report: JXTA: A Network Programming Environment", *IEEE Internet Computing*, volume 5, 2001

[19] JXTA, Web Site, http://www.jxta.org/

[20] Majitek, web site, http://www.majitek.com

[21] Bieber, G, Carpenter, J., "Introduction to Service-Oriented Programming (Rev 2.1)", *online document*, April 2001. (http://www.openwings.org/download.html)

[22] Chen, K., Gong, L., "Programming Open Service Gateways with Java Embedded Server Technology", *Publ. Addison Wesley, August 2001, ISBN 0201711028.*

[23] Mobilet, Web Site, http://www-adele.imag.fr/~desertot/dev/osgi/mobilet/mobilet.htm

[24] R.S. Hall and H. Cervantes, "An OSGi Implementation and Experience Report," IEEE Consumer Communications & Networking Conference (CCNC), January 2004

[25] OBR, Oscar Bundle Repository, Web Site, http://oscar-osgi.sourceforge.net/