Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 10 (18): 2154-2160, 2010
ISSN 1812-5654

Interoperable Mobile Agent with JXTA

R. Mekki
USTO M.Boudiaf, B.P. 1505, ElMnaouer Oran, Algeria

Abstract: As the web continues to grow in both content and the number of connected devices, peer-to-peer computing is becoming increasingly prevalent. By using peer to peer, applications could provide access to a variety of resources with high availability at a lower cost. We use peer to peer technology to deploy mobile agent community. Their objective is to gather data about local state of machine and to build more abstract observation of the whole network. The peer to peer configuration is defined by the use of JXTA framework and the mobile agent can travel from one machine to another by the use of XML serialization. Because this kind of serialization allows keeping state of agent, data collecting is realized with a community of mobile agents, each of them are responsible of a type of data.

Key words: Mobile agent, peer to peer, data collection, application domain, software inventory

INTRODUCTION

Agents are meant to work with other agents. A key feature of the agent paradigm of software development is that communities of agents are much more powerful than any individual agent, which immediately highlights the necessity for interoperable agent systems. But the mobile agent community and the agent community do not fill the same need. They do not concern about the same problems when talking about interoperability amongst agents.

For the mobile agents community interoperability work focuses on the execution environment and the standardization of some of its facets and features; in the case of non-mobile agents, there is no notion of an execution environment and the focus is only on communication as the means for achieving interoperability. In the latter case, interoperability is similar to effectively exchanging the data and knowledge content of the agents.

Mobile agents migrate to an agent host where an execution environment is set up for them; upon arriving there, they might execute task or their own code, make remote procedure calls in order to access the resources of the host, collect data and eventually might launch another process of migration to another host. While residing on a particular agent host, mobile agents might have limited interaction (communication) with other agents on the host through an RPC-type mechanism. A potential problem arises from the fact that not all platforms for mobile agents are the same. Moreover, the network between two nodes does not support the same protocols. Depending on security tools, some protocols are available and some others are not. Our approach is an answer to the loss of interoperability not only for the agent host and the data exchange but also for the protocol used for the data transfer.

First we describe the domain of mobile agent community and their standard. Then we propose technical choices for description of the map between the computers but also for the implementation of agent host and mobile agent. Next, we detail an example where some technical aspects are shown to highlight the robustness of the hypothesis. Finally, we sum up regarding the first results and the next step is depicted.

A MOBILE AGENT SURVEY

Few standards exist concerning mobile agents. The Mobile Agent Facility (MAF) proposal was subsequently replaced by MASIF (Milojicic *et al*., 1999, 1998). It is a first tentative to standardize some aspects of this execution environment. The proposal was still under investigation by the Object Management Group (OMG), as of the mid-1998. MASIF is a collection of definitions and interfaces that provides an interoperable interface for mobile agent systems.

The MASIF's interoperability is not only about language interoperability, such that language based on virtual machine but instead it aims at interoperability between agent systems written in the same language although possibly by different vendors. MASIF focuses on standardizing three things: agent management, agent transfer and name convention.

The agent management means the standard operations such as creating an agent, suspending it, resuming, and terminate it. The agent transfer describes a common infrastructure for agent applications to freely

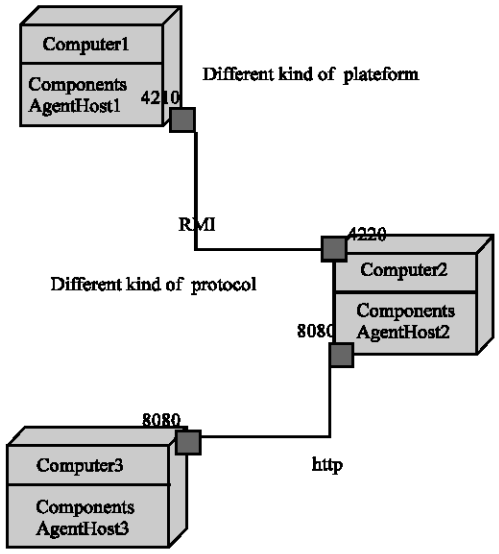

Fig. 1: UML deployment diagram. Note that each node is representative of local network (a) specific computer system). It supports specific kind of code (constraints: language, communication, security rules, etc.)

move among agent systems of different types. Finally name convention is essential for agents and agent systems. It is expected that the use of a standardized syntax and semantics of agent and agent system names will allow agent systems and agents to identify each other, as well as clients to identify agents and agent systems. This specification has an obvious constraint in a real context: it is not possible to insure that every node of the network contains a mobile agent receiver written in the same language.

Also, the new challenge is to consider not only a local network but also wider network which includes information systems from several companies. It means that each information system has its own constraints such that: software management, programming language, communication port, income protocol, outcome protocol, etc. On Fig. 1, three distinct computers are present; each of them plays a server role in more complex software architecture. If a traditional approach is adopted, the same protocol has to be chosen for all agent migration, such instance JINI in Cheiron project (Gorissen *et al*., 2006). More often, the same language is imposed for all agent host and mobile agent. It involves that all data exchange are realized through the same data type; this is the context of GrassHopper developments (Baumer and Magedanz, 1999).

Today, these constraints are unacceptable and the component called AgentHost1 can be written in C# programming language and receives a mobile agent, initially developed in Java programming language. This data exchange can be done only if the data are encoded into a standard between both programming languages. It is commonly admitted that XML is a well formed language, especially for the object serialization. This technical choice is the basis of web service call and the SOAP language which can be used for calling any remote object.

Another limit is the vehicle used by the mobile agent for the migration. If the component called AgentHost2 receives a mobile agent on RMI protocol, this is because this protocol is available as an income protocol for that computer. But it is surprising that it obliges the mobile agent to leave this agent host through the same protocol. If http protocol is the only available protocol with the computer called Computer3, then http has to be used independently from the other protocol. This constraint was one of the most limitative one in works referred (Bernichi and Mourlin, 2006; Dumont *et al*., 2007). Instead of using the same protocol everywhere, it is more interesting to use the most powerful where it is possible. This is why a protocol like JINI is not enough used in mobile agent programming even if it contains a large set of features. We adopted a peer-to-peer (P2P) protocols that allow any networked device communicate and collaborate mutually as peers. This protocol, called JXTA, can inter connect sensors, cell phones, PDAs, laptops, workstations, servers and supercomputers. Also, it insures a large domain of application (Arora *et al*., 2002).

TECHNICAL APPROACH AND NETWORK DESCRIPTION

The JXTA protocols are programming language independent, and multiple implementations, also known as bindings, exist for different environments. Their common use of the JXTA protocols means that they are all fully interoperable.

This technology is designed to enable peers provisioning P2P services to locate and communicate with one another independent of network addressing and physical protocols. It is also designed to be independent of programming languages, network transport protocols, and deployment platforms.

The JXTA provides a common set of open protocols backed with open source reference implementations for developing peer-to-peer applications. The JXTA protocols standardize the manner in which peers: discover each other, self-organize into peer groups, advertise and discover network resources, communicate with each other, monitor each other.

Fig. 2: The JXTA network consists of a series of interconnected nodes, or peers. Three kinds of layers are used (JXTA core layer, service layer, security layer)

The protocols can be implemented in the Java programming language, C/C++, .NET, Ruby, and numerous other languages. Furthermore, they can be implemented on top of TCP/IP, HTTP, Bluetooth, and other network transports all the while maintaining global interoperability (Traversat *et al*., 2003).

The JXTA protocols enabled us to build and deploy interoperable agent and agent host. Because the protocols are independent of both programming language and transport protocols, heterogeneous devices with completely different software stacks can interoperate with one another. Using JXTA technology, we can write networked, interoperable applications that can: find other agents on the network with dynamic discovery across firewalls and NATs, share resources with any agents across the network, find up to the available content at agent hosts, create a community of agents that provide a service, securely communicate with other peers on the network.

Figure 2 shows a real description of material which was schematically represented on Fig. 1. The mobile phone sends a mobile agent. It contains a set of physical measures via Bluetooth to a laptop where the data are filtered. When all the data are collected by the laptop, it packages all of them and sends them by the use of another mobile agent through a local protocol.

AGENT DESCRIPTION

Each agent host provides a set of services and resources which it makes available to other agent host. Mobile agents are interactive programs and can include authentication systems, data filter or almost any program that can be networked.

A mobile agent's resources are normally static or non-interactive content which the mobile agent either controls owns or even merely has a copy of. Resources can include files, documents, media, advertisements, indexes but can also include real world resources such as switches, sensors and printers.

Agents can organize themselves into agent community or agency. An agent community, loosely defined, is any set of agents that provision and leverage a common set of services for a common purpose. There are two key aspects to this definition: common services and common purpose. Two agent communities might have the same set of services, for example a chat application, but different purposes, for example politics chat and sports chat.

The JXTA agent hosts use sockets and pipes (Mekki and Fezza, 2009) to send messages to one another. Sockets are reliable bi-directional connections used for applications to communicate reliably. Pipes are an asynchronous and unidirectional message transfer mechanism used for service communication. Messages are simple XML documents or data whose envelope contains routing, digest, and credential information. Pipes are bound to specific endpoints, such as a TCP port and associated IP address.

Four essential aspects of the JXTA architecture distinguish it from other distributed network models like JINI. First, the use of XML documents (advertisements) to describe network resources. Second, there is an abstraction of pipes to agents, and agents to endpoints, without reliance upon a central naming/addressing authority such as DNS. Third, a logical addressing id built and finally a decentralized search infrastructure is based on Distributed Hash Table (DHT) for resource indexing.

NETWORK ARCHITECTURE

The network is an Ad-Hoc and adaptive network composed of connected agent hosts. Connections in the network may be transient and, as a result, message routing between agent hosts is non-deterministic. Agent hosts may join or leave the network at any time; which results in ever changing routing information.

The only common aspect that various JXTA applications share is that they communicate using JXTA protocols. The organization of the network is not mandated by the JXTA framework, we developed four kinds of agent hosts.

Minimal agent host: A minimal agent host can send and receive messages or serialized mobile agent, but does not cache advertisements or route messages f or other agent host. Agent hosts on devices with limited resources (e.g., a PDA or cell phone) would likely be minimal agent hosts.

Full-featured agent host: A full-featured agent host can send and receive messages and will typically cache advertisements. A simple agent host replies to discovery requests with information found in its cached advertisements, but it does not forward any discovery requests. In any JXTA deployment most peers are likely to be agent hosts.

Rendezvous agent host: A rendezvous agent host is an infrastructure host, it aids other agent host with message propagation, discovery of advertisements and routes, and most importantly it maintains a topology map of other infrastructure hosts, which then used for controlled propagation, and maintenance of the distributed hash table. Each agent community maintains its own set of rendezvous agent hosts and may have as many rendezvous agent hosts as needed. Agent hosts send search and discovery requests to their rendezvous agent host which in turn may forward requests it cannot answer to other known rendezvous agent host using the topology mapped distributed hash table.

Relay agent host: A relay agent host s an infrastructure, it aids non addressable (firewalled/NAT'd) hosts with message relaying. An agent host may request an in memory message box from a relay agent host to facilitate message relaying whenever needed.

If we apply this design to our earlier example, Fig. 3 is a collaboration diagram where we observe le locality of the two first computers. First, all agent host are instance of a class called FullFeaturedAgentHost. But because the server is protected with a firewall and other security tool, we have to adapt the mobile agent exchange. Because the instance "agentHost2" wants to set a synchronous exchange, an instance of RendezVousAgentHost class has to check the emission and the knowledge reception. Then an instance of RelayAgentHost has in charge to select the right protocol to export mobile agent through the firewall to agenthost3 instance. The label on each message is used to express the causality of events.

An agent host behind a firewall can send a message directly to an agent host outside a firewall, but an agent host outside the firewall cannot establish a direct connection with an agent host behind the firewall. The same is true for agent host which are behind a NAT device.

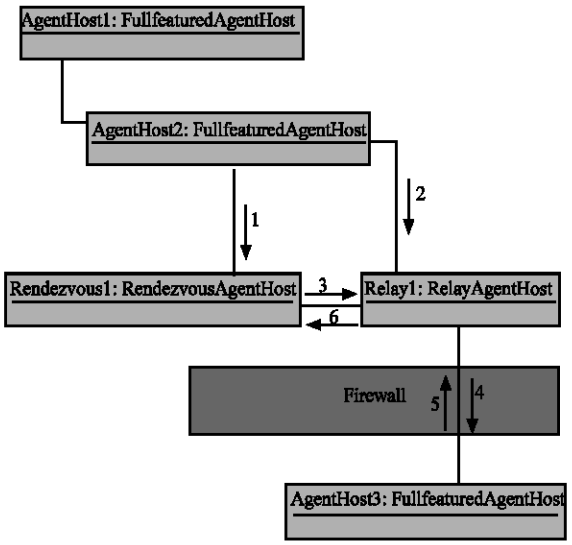

Fig. 3: UML collaboration diagram. Note that message routing scenario across a firewall limits the data exchange, especially for mobile agent navigation

In order for agent hosts to communicate with each other across a firewall, three conditions exist. First, at least one agent host in the agent community inside the firewall must be aware of at least one agent host outside of the firewall. Secondly, the agent host inside and the agent host outside the firewall must be aware of each other and must support a common transport (HTTP or TCP). Finally, the firewall, at the very least, has to allow outbound HTTP or TCP connections.

MOBILE AGENT ARCHITECTURE

We choice a class structure which respects a design pattern which was presented in a previous work (Bernichi and Mourlin, 2007). We updated that work by the use of dynamic tasks which allow the mobile agent to change its task during it migration over the network. The Task interface has a code constraint which impose to develop a perform method with a mapping parameter. This one is essential for the evolution of the task. In our example, a mobile agent migrates through three nodes (or agent host), but the task depends on the kind of agent host. Because the order of the agent host is not necessary a constant of its travel, we chose a work flow where each step in the travel of a mobile agent decides of the following step. That approach seems us to be more adaptative.

The classes called Receiver and Sender are threads which allow a mobile agent to import or to export some other mobile agent (Fig. 4).

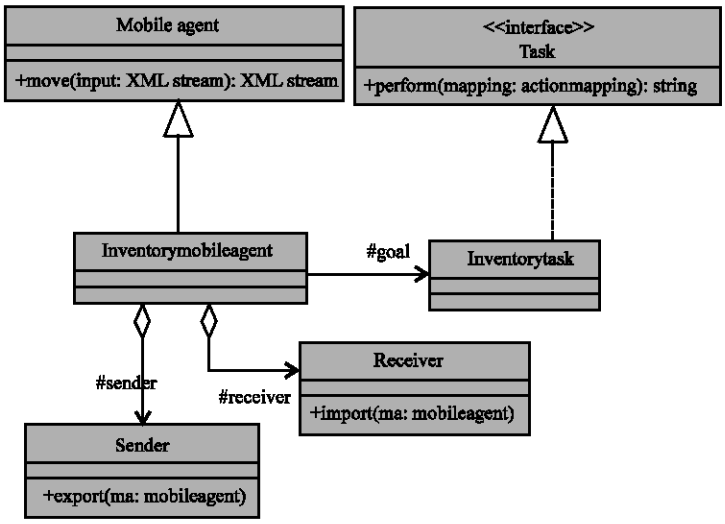

Fig. 4: UML class diagram. (Partial view of a class diagram): mobile agent design pattern is applied

APPLICATION DOMAIN: SOFTWARE INVENTORY

We can perform a manual inventory by checking the hard drive of each PC of our network and recording the information in a report. One easy way is to start with the first PC and to view the Add or Remove Programs screen on each PC. This activity seems to be boring and mistakes are easy to do. Also, we can use a software inventory tool to perform an automatic inventory of your company's PCS and servers. This can be done by a mobile agent community where each of them has a set of agent host where it has to walk through.

The community will scan PCS that are on a network or to scan each non-networked PC (stand-alone) and then generate software inventory reports. If a mobile agent is on stand-alone PCS, then a mobile agent will be needed to install the tool on each PC. When the community inventoried all of your company's PCS, combine the information into one master report.

For our case study three Task class are developed according to the platform.

- **The first class is called software details for the cell phone:** It provides details of the commercial / non-commercial software used in the network with details like Vendor Name, Installation Date, Software Version, etc.
- **The second class is called software metering for the laptop:** It views the software usage details in each computer like, rarely used, occasionally used, or, frequently used
- **The thirds class is called software license compliance for the server:** It provides the compliant (including over-licensed software) and non-compliant (under-licensed) software used in the network.

The tasks are published into a lookup service where the mobile agent can look for them.

Since, JXTA defines a series of XML messages for communication between peers, we use these kinds of XML messages for the administration of the JXTA network.

In our context, for each mobile agent class, we can build an XML schema and then by the use of JAXB framework (Toth and Valenta, 2006), the instances of that class are serialized into XML stream and the link with their XML schema is kept. Also, after the reception, it is possible to check the contents of the XML stream before the de-serialization. Also, if the stream respects its schema, the receiver agent host binds a schema; it means generating a set of Java classes that represents the schema. Then, for each mobile agent class, three classes are developed and used by the JAXB compiler. First, an unmarshaller agent class governs the process of deserializing XML data into Java content trees, optionally validating the XML data as it is unmarshalled. Secondly, a marshaller agent class governs the process of serializing Java content trees back into XML data. Finally, a validator agent class: performs the validation on an in-memory object graph. Of course, the programming language depends on the chosen language for the agent host.

Local activity: When a mobile agent executes its task suite, the trace of its activity is saved in a local file. In our example, the first node is a cell phone. It receives XML

```
Local configuration found
Loading found configuration
Configuration loaded
Starting JXTA
JXTA Started
Peer name : computer1
Peer Group name: NetPeerGroup
PeerGroupID:urn:jxta:uuid-
12201252656162614E50472050354E4DE5445D2F685244AEBCB98AA03662779603
Waiting for a rendezvous connection for 45 seconds (maximum)
Connected :true
Stopping JXTA
```

Fig. 5: A look at the log file highlights the main steps

```
Sending a Discovery Message
Sending a Discovery Message
Got a Discovery Response [2 elements] from peer : computer1
Peer name = computer2
Peer name = computer3
Sending a Discovery message
Got a Discovery Response [2 elements] from peer : computer1
Marshalling InventoryAgent
Export computer2
```

Fig. 6: Discovery of nodes

```
Starting JXTA
reading in socket.adv
Connecting to the server
Reading in data
received 579 bytes
Sending back 65536 * 1824 bytes
Completed in :28673 msec
Data Rate :44089 Kbit/sec
Connecting to the server
Reading in data
received 579 bytes
Sending back 65536 * 1824 bytes
Completed in :14743 msec
Data Rate :63344 Kbit/sec
Validating InventoryAgent stream
Marshalling InventoryAgent class
Import computer3
```

Fig. 7: Reading XML streams

stream via Bluetooth, apply the associated unmarshaller class and perform the control onto a list of applications. All the results are saved in the private part of the agent; also these data will be serialized by the end of its activity (Fig. 5).

Agent export: When the task suite is ended, the mobile agent notifies its agent host and that one exports the mobile agent into its lookup service and then it will continue its visit of the network. To do that, the agent host has to use the marshaller class related to the mobile agent class. This XML stream is validated with the associated XML schema. This is due to the changes of the state of the mobile agent.

Then, this valid XML stream is sent to a lookup service where the stream is transformed into an object. This first migration is quite simple because the JXTA architecture contains only 2 peers (Fig. 6). The trace will be complex with a relay.

Agent import: The agent host decides the reception of mobile agent. Thus, it can precise that only one mobile agent can be present, or more precisely the multiplicity per class. It requests its lookup service to know if the mobile agent is available. This request is about the name and the signature of the operation of the mobile agent.

The agent host receives an XML stream and executes reverse strategy. First, it uses the Validator class associated to that class of agent. Then, if it is correct, it calls the unmarshaller class to obtain an object (Fig. 7).

The lookup service has its own management strategy. Also, when an agent host receives a XML stream, that one is no longer available with that lookup service. It will be available into another lookup service by the end of its activity with that agent host.

DISCUSSION

When the roadmap is finished, the last agent host receives mobile agent for data extraction and the creation of the final report. Because three different tasks were done on different agent hosts, the final result is an aggregation of each result.

Because periodically this activity is realized, the current collect is appended to the previous software details and agent host stores them in a database. The inventory scanning interval is flexible and can be

configured to meet the real-time needs. This enables administrators to get up-to-date inventory information at any given time without any manual intervention.

If information has to be added for the next inventory, new tasks have to de created and added to the task suite for an agent host.

For instance, when new software is detected in agent host, an event is generated and its trace is saved. Again, when a non compliance (under-licensed) of software licensing policy, i.e., the license is inadequate and has to purchase more licenses to be compliant, another event is generated into the private part of the mobile agent.

When prohibited software is detected in agent host, this allows the administrator isolate this node from the others.

This allows us that our toolkit of mobile agents is able to cover more devices and several kinds of protocols, whereas this is the most limitative constraint in earlier study (Bernichi and Mourlin, 2006; Dumont *et al*., 2007).

CONCLUSION

We detailed a short example about inventory. This case study is available to everyone for a better understanding of present study. We also added security rules to agent host to authenticate mobile agent and to provide specific permission. We do not how protection domains are managed but this is the realm of agent host.

The next step is to combine the use of JXTA protocol with a previous work with JINI mobile agent and to observe the possible conflicts. We will try to prove that the JXTA implementation is able to load more mobile agent on an agent host. This load is a key concept in numerical domain.

REFERENCES

Arora, A., C. Haywood and K.S. Pabla, 2002. JXTA for J2ME™, extending the reach of wirless with JXTA technology. http://pegasus.javeriana.edu.co/~mad/JXTA4J2ME.pdf.

Baumer, C. and T. Magedanz, 1999. The Grasshopper Mobile Agent Platform Enabling Shortterm Active Broadband Intelligent Network Implementation. In: Lecture Notes in Computer Science, Covaci, S. (Ed.). Vol. 1653, Springer, Berlin, Heidelberg, ISBN: 13-978-3-540-66238-9, pp: 109-116.

Bernichi, M. and F. Mourlin, 2006. Mobile agent communication scheme. Proceedings of the International Conference on Systems and Networks Communications, Oct. 29-Nov. 03, IEEE Computer Society Press, Tahiti, French Polynesia, pp: 6-6.

Bernichi, M. and F. Mourlin, 2007. Software management based on mobile agents. Proceedings of the 2nd International Conference on Systems and Networks Communications, August 2007, IEEE Computer Society Press, Cap Esterel, France, pp: 64-64.

Dumont, C., F. Mourlin and A. Mobile, 2007. Computing architecture for numerical simulation. Proceedings of International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Nov. 4-9, Papeete, French Polynesia, pp: 68-74.

Gorissen, D., P. Wendykier, D. Kurzyniec and V. Sunderam, 2006. Integrating heterogeneous information services using JNDI. Proceeding of the 15th HCW 2006 Heterogeneous Computing Workshop, April 2006, Rhodes Island, Greece, pp: 1-10.

Mekki, R. and R. Fezza, 2009. A sample chat application based on JXTA. J. Applied Sci., 9: 3912-3916.

Milojicic, D., M. Breugst, I. Busse, J. Campbell and S. Covaci *et al*., 1998. Mobile agent system interoperability facility. Proceedings of the 2nd International Workshop on Mobile Agents, LNCS. 1477, (IWMA'98), Springer-Verlag, pp: 50-50.

Milojicic, D., M. Breugst, I. Busse, J. Campbell and S. Covaci *et al*., 1999. The omg mobile agent system interoperability facility. Proceedings of the Personal Technologies in ACM, December 1999, Addison-Wesley, pp: 628-641.

Toth, D. and M. Valenta, 2006. Using object oriented technologies for native XML database systems. Departmetn of Computer Science and Engineering FEE, CTU in Prague Czech Republic, Vol. 176. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-176/pres3.pdf.

Traversat, B., M. Abdelaziz, M. Duigou, J.C. Hugly, E. Pouyoul and B. Yeager, 2003. Project JXTA Virtual Network. Sun Microsystems, UK.