Proceedings of the 4th National Conference; INDIACom-2010
Computing For Nation Development, February 25 – 26, 2010
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

A Practical Approach towards Development of Applications and Solutions using Java Agents

Nilesh M. Shelke[(1)], Dr. Rajiv Dharaskar [(2)]
1. GHRCE, CRPF Gate 3, Digdoh Hills, Hingna Road, Nagpur
2. Professor, CSE Deptt. GHRCE, Nagpur

EXTENDED ABSTRACT
Abstract: Agent technology is widely spread on the software domain and agents can be found in a variety of applications and solutions. Java agents are agents written in the Java™ programming language. Java is considered the perfect language for writing agent solutions.

At the beginning this paper close look is given at agents: what they are, where they are and how and why they work. After agent technology and Java agents are introduced, benefits and drawbacks are presented. Focused on pros and cons of agents the agent technology is compared to other technologies. Real world applications and solutions are introduced. In addition, problems that agent technology can solve are covered. A look into the future shows possible branches where agents will boom in the future. Development environments are needed to adapt agent technology to the office of everyday computer workers. Some common development tools are presented.

KEYWORDS: Java, Agents, Software, Mobile Agents

I. INTRODUCTION

The advent of software gave rise to much discussion of just what an *agent* is, and of how agents differ from programs in general. Every problem seemed to have an easy solution based on *agent technology*.

Following is the organization of the paper. Chapter 2 explains the basics of agents and introduces Java as agent language. Chapter 3 is devoted to the advantages and disadvantages of Java agents. In addition, comparisons to other technologies are made. Chapter 4 presents applications and solutions based on Java agent, real-world samples and a look at future domains of agent technology. Chapter 5 deals with development tools and platforms for Java agents.

II. WHAT IS AN AGENT?

The word agent has found its way into a number of technologies and it is a branch on the tree of *distributed computing*. Indeed, the word agent is well known and everybody can link it to figures like James Bond. However, *digital agents* are relatively unknown, even by people on the IT territory.

Software agents can find information of importance in the dark forest of junk. These agents can filter unimportant messages out of the incoming e-mail, surf the web with the purpose to find information on a specific issue or buy a cheap weekend-trip to Paris by gathering data from different online-travel agencies and comparing their prices. In short, agents are ready to do peoples dirty work. Once the agent has got its mission, people can lean back and concentrate on other things while the agent is fulfilling its goal. Of course, agents can do a lot more than only the tasks mentioned above. These days especially the branch of *e-commerce* is interested on solutions based on agent technology.

It is not easy to find a definition covering all things that agents can be considered to be and excluding all of the things they are not. Franklin and Graesser believes [4,5] that many definitions made by workers involved in agent research grew directly out of the set of examples of agents that the definer had in mind. A crucial problem is that there exists no standardized definition for software agents.

To compare software agents to a real world model on an abstract level, a human being can be seen as an agent. "An *autonomous agent* is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. A human is an agent with multiple, conflicting drives, multiple senses, multiple possible actions and complex sophisticated control structures." [4, 5]

To compare software agents to a real world model on an abstract level, a human being can be seen as an agent. "An *autonomous agent* is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. A human is an agent with multiple, conflicting drives, multiple senses, multiple possible actions and complex sophisticated control structures." [4,5]

Agents can be classified with help of *characteristics* (*attributes, properties*) they typically possess. Other classification schemes, such as e.g. classify agents according to the tasks they perform, are treated in [4,5], [11] and [12].

A. Agent architecture

The following presents the architecture of an agent. This agent possesses following characteristics: mobility and persistence, communication and collaboration, and adaptation, learning and goal orientation.

Sundsted presents [12,13,14] following key requirements that such agent architecture must satisfy:

1. An agent must have its own unique identity
2. An agent host must allow multiple agents to co-exist and execute simultaneously
3. Agents must be able to determine what other agents are executing in the agent host.
4. Agents must be able to determine what messages other agents can accept and send.
5. An agent host must allow agents to communicate with each other and the agent host.
6. An agent host must be able to negotiate the exchange of agents
7. An agent host must be able to freeze an executing agent and transfer it to another host
8. An agent host must be able to thaw an agent transferred from another and allow it to resume execution.
9. The agent host must prevent agents from directly interfering with each other

For realizing the architecture above, clearly an *Agent class* and an *AgentHost class* are needed. Because one of the key requirements (number 9 in list) refuses agents from interfere directly with other agents also an *AgentInterface class* is needed. With help of the AgentInterface class, other agents' public methods can be invoked. Finally yet importantly, the first key requirement in the list demands an *AgentIdentity class* that identifies agents (both to themselves and to others) and allows them to decide if they want to accept or discard messages from other agents. [13,14]

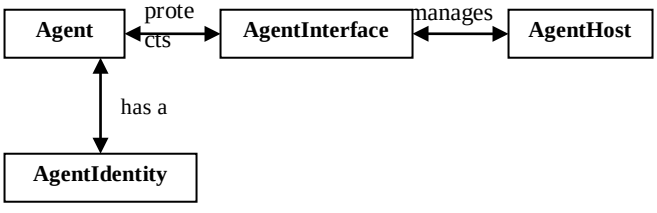

Figure 2.1: sample agent architecture

Figure 2.1 shows how the different classes belong and work together. To get a better idea of the classes the following list explains their purposes [13,14]:

- **AgentIdentity:** As mentioned earlier, every agent has an identity. The agentIdentity class defines such a unique identity for each agent, which then can be used in collaboration with others to decide with what agents it wants to communicate.
- **Agent:** The agent class defines the agent. For every agent executing on an agent host there exist an instance of the agent class.
- **AgentInterface:** An instance of the AgentInterface provides access to other agents (or vice versa) via a well-defined interface. Agents do not directly interfere with others; they interfere via an agent interface.
- **AgentHost:** The AgentHost defines an agent host. With help of an instance of the AgentHost, hosts can work together and transfer agents. An AgentHost instance knows exactly what agents are executing in the system.

B. Agents and Java Agents

A Java Agent differs from an Agent in the way that the agent is based on Java technology. In other words, the agent is written in the *Java programming language*. Java Agents implement the classifications and characteristics as well as of course, the behavior of agents. The definitions and explanations above surely match to Java Agents. In fact, many things can be realized by implementing them in Java: an agent written in another language could hardly match what Java agents can achieve.

Java makes the perfect language for agents. The pros of Java as agent language are the *platform independence*, *network-centricity* and the so-called *sandbox security model* [11]. Due to these benefits, the language has been used preferably to develop agent-based tools (such as e.g. IBM's aglets, *JATLite* etc.).

Because the *Java RMI* (Remote Method Invocation) and several implementations of the *CORBA* (Common Object Request Broker Architecture) specification are available to Java programmers, the use of *distributed objects* is no problem.

Methods on objects residing on other computers can be invoked as easily as invoking methods on local objects [14].

C. Agents vs. objects

An object is a combination of *data structures* and their corresponding methods, also called functions. Objects are used for abstraction of passive entities in the real world: for example a car with its color, type of tires, brand etc.

Agents have a lot in common with objects. Tveit describes [15] agents to be "regarded as possible successors of objects since they can improve the abstractions of active entities". Agents also support structures for representing mental components such as simple beliefs and commitments: e.g., choices can be made on ways known from the domain of artificial intelligence.

D. Agent-Orientated Programming, AOP

Agent-Orientated Programming, shortly AOP, is considered a specialization and extension of *OOP* (Object-Orientated Programming). In OOP, systems consist of objects communicating with other objects to perform internal computations. AOP improves OOP in having agents instead of objects. The interaction between agents works by sending messages adapted from the "*speech act*" theory. Internal computations performed by agents are based on beliefs, capabilities and choices [5].

III. PROS AND CONS OF JAVA AGENTS

Several specialists on the agent-technology domain try [10] to calm down over-enthusiastic views that agent technology would be perfect and everything could be solved with agents. It is about finding and creating better solutions based on agent technology and nothing else, and every solution has its pros and contras.

Because of the fact that agent technology can and does solve several problems better and easier than other technologies, some of the major competitors of agents are presented below. However, to see all of the pros and contras, not only the benefits of agent technology compared to other technologies, they also need to be treated separately. By examining the general advantages and disadvantages, one can decide whether to built solutions with agents or not.

3.1 Java Agents vs. other software technologies

Problems can be solved with all kind of solutions and goals can be achieved on many different ways. Java Agents can put other software technologies into the shadow. Then again, other technologies may let Java Agents look poor in some situations. When and why to decide for a solution based on Java Agent technology is documented in the following.

3.1.1 Typical client/server applications

The pros of Java Agents come to daylight especially when comparing them to typical client/server applications. Client/server applications work fine, no doubt about that. However, in some situations or environments client/server applications work only on a very unreliable or/and inefficient level.

The *bandwidth* problem is well known. Network bandwidth is valuable and may be limited, which limits automatically the interest in client/server applications: A transaction between client and server usually requires many round trips, handshakes, acknowledgements of success and of course, the actual data transfer all consumes bandwidth. If there are many transactions between clients and/or servers simultaneously, the required bandwidth may exceed what is available. Figure 3.1 shows the communication between client and server.

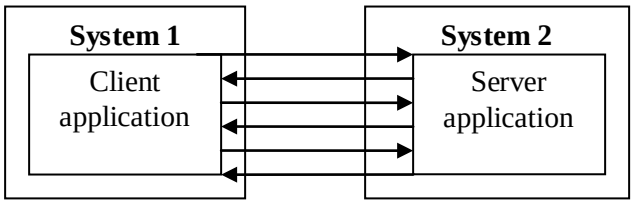

Figure 3.1: Client/server application communication with many round trips

Java Agents bring a solution to the nagging problem with bandwidth: An agent can be created and sent from the client to the server and returned after it has fulfilled its "mission". This way bandwidth consumption is reduced because only the agent need to be sent to the server, where it does the same things a client/server application would do over the wire. This reduction of bandwidth use is presented in figure 3.2.

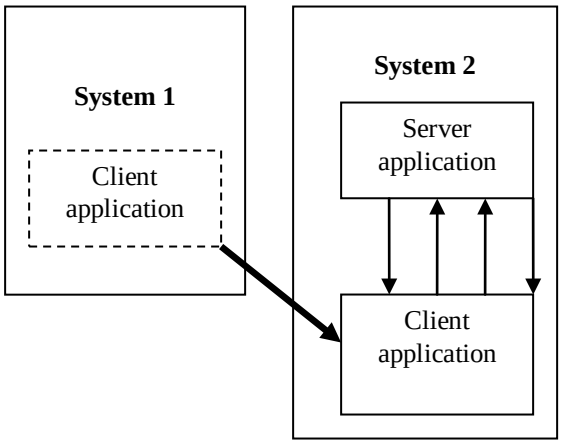

Figure 3.2: A mobile agent can be sent to the server where requests are made directly, not over the wire.

Another problem of client/server applications is that the network connection has to be alive during the whole transaction or query. If the connection breaks down the client usually needs to restart the transaction or query from the beginning. If the connection just slows down the whole process suffers from that "speed limit". Using agent technology, the Java Agent can be sent to the server and the connection may die or be shut down manually (e.g. for saving money if using a *dial-up* connection). The agent handles the transaction or query on its own and the result will be presented to the client when it gets online again.

Yet another benefit of Java Agents is that the system can be easily modified after it is built. During design time, fewer decisions need to be made than with client/server applications. When wrong decisions are made, the client/server-system is doomed to suffer and afterwards it is difficult if not even impossible to fix problems. Sundsted reminds [[12,13,14]] that "the agent technology even allows changes to be made after the system is built and in operation".

3.1.2 Applets

For comparing *applets* with Java Agents, IBM's Aglets are used to represent the Java Agent technology. Venners define [16,17]. Aglets as "a mobile-agent technology built on top of Java".

An applet is code that can move across a network from a server to a client. Aglets are an extension to applets because they not only carry the class file (the code) like applets but also their state. Thus, an aglet is a running Java program that can move on a network, from one host to another. With help of the state an aglet carries, it can migrate sequentially on many destinations and even return to the host where its journey began [16,17].

IV APPLICATIONS AND SOLUTIONS BUILT ON JAVA AGENTS

4.1 Example tasks for agents

A list of potential tasks for Java Agents is provided in [16,17]. The example tasks mentioned below are only a fraction of what can be done with agent technology.

Distributed data collection: Mobile agents can be used for applications collecting data spread across a network. Because mobile agents can travel sequentially to many sites, they can collect distributed information and return to the point of origin to make a report.

Searching and filtering: The Internet offers an incredible amount of information of which a huge part can be considered as garbage for one searching for specific data. One can give mobile agents a search criterion for visiting web sites and building an index of links to sites matching that criterion. Searches and filtering by agents can save a lot of time.

Monitoring: Information can also be spread across time when it is produced constantly and published on the network. In this kind of situations agents do not need to travel across the network (of course, they can if necessary) but they can wait for certain kinds of information to become available. Venners mention [16] an example agent that could go to a stock market host and wait for a certain stock to reach the desired price. On behalf of its user, it the agent could then buy some of that stock.

Targeted information dissemination: With help of agent technology, interactive news of advertising to interested parties can be distributed. This advantage can yet uncover another drawback of agents: they can be used to spam people via e-mail.

Negotiating: Agents can gain information by interacting with other agents. Again, Venners offers [17] an interesting example of such a task: "If you want to schedule a meeting with several other people, you could send a mobile agent to interact with the representative agents of each of the people you want to invite to your meeting. The agents could negotiate and establish a meeting time."

Shopping: Electronic commerce is a well-suited environment for agent technology. An agent can be sent to the Internet to make orders and potentially even pay whatever a user needs to buy. The agent can search information on a certain article, compare prices from different vendors and even pay the article using the user's credit card number. E-commerce can also take place between two (or more) agents: one agent wants to sell a product and the other is interested in buying one.

Entertainment: Agents can represent game players and compete with others on a game host. Online games can also be played with real money: again, by using a user's credit card number online casinos can make a user happy or ruin him or her totally.

Viruses: Of course, agents can also be destructive. If an agent gets access to local resources on agent hosts, virus writers and rogue programmers could have a lot of fun. Agents can have other malicious behaviors, too: agents popping from host to host with the purpose to eat as much bandwidth capacity as possible (if not all) can be considered as some kind of virus, too.

4.2 A real world solution

Gossip is a widely deployed application using Java agent technology. In the year 2000 more than 25'000 people in the Netherlands were using Gossip. The idea of Gossip is simply to

provide answers to questions. Lawton describes [9] Gossip as an "application that provides an environment where Java experts can pose questions and post lists of their favorite answers". Given a question, the agent asks others for answers and returns the results. Answers can be rated by quality that agents can find the agents providing the most relevant information.

4.3 Future domains of agent technology

Even if agent technology can be used in a wide range of applications, some branches are often considered of growing to forthcoming major domains. Interviews with industry experts show [10] that in the future, interest will focus especially on *network quality service management* and how to manage different qualities of service.

E-commerce is also clearly a major area where agent technology will be implemented widely. By using agents in electronic commerce, it can become more automated and also more flexible and dynamic. Greenwood's opinion is [6] that "agent technologies will have a significant role in the evolving online marketplace."

Making commercial decisions can be placed in the hands of agents. Commercial transactions can be made by selling and buying agents. Agents offer a number of useful possibilities in e-commerce.

4.4 Sample source code

For introducing some Java agent code, this section contains a simple implementation of cloning Java agent. The example is presented by Sundsted [12,13].

```
abstract public class Agent
public class CloneLimiter implements CloneListener()
{
        final integer MAX = 5;
        boolean original = true;
        int number_of_clones = 0;
// Called when the aglet is about to be cloned.

public void onCloning(CloneEvent ev)
{
if (original == false)
throw new SecurityException("Clone cannot create a clone");
if (number_of_clones > MAX)
throw new SecurityException("Exceeds the limit");
}
// Called in the cloned aglet.
public void onClone(CloneEvent ev) {
original = false;
}
// Called in the original aglet after the cloning.
public void onCloned(CloneEvent ev)
{
number_of_clone++;
}
}
```

For an extension of the example and the rest of the code needed for the implementation (AgentHost class and AgentHostImplementation class). The example implementation relies on RMI to provide transparent network communication between agent hosts, and on object serialization to transfer the agent.

IV. DEVELOPMENT TOOLS

For fast and effective development and maintenance of agent solutions, comprehensive tools are required [3]. Development tools must not only be usable by Java developers but also by customers.

Brantschen and Haas mention [1] that object (-orientated) technologies were not that favored until robust platforms and tools became available. The same phenomenon applies to agent (-orientated) technologies.

A wide range of different development tools exists. Some well-known development tools and platforms for Java agents are IBM's Aglets and JATLite.

5.1 IBM's Aglets

Aglets are a mobile-agent technology built on top of Java. An aglet is a Java object that can move from one host to the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution on the "new" host.

Aglets are similar to applets in that they run as threads inside the context of a *host Java application.* Aglets need a host *Java application* to be running on a computer they want to immigrate. The host application is called an "*aglet host*". The aglet host installs a *security manager* to enforce restrictions on the activities of any untrusted aglet.

5.2 JATLite

"JATLite (*Java Agent Template, Lite*) is a package of programs written in the Java language that allow users to quickly create new software "agents" that communicate robustly over the internet. JATLite also provides a basic infrastructure in which agents register with an *AMR* (Agent Message Router) using a name and password, connect/disconnect from the Internet, send and receive messages, transfer files with *FTP* (File Transfer Protocol), and generally exchange information with other agent on the various computers where they are running." [7]

With JATLite, it is easy to build systems in a common way because it provides a set of *Java templates* and a ubiquitous Java agent infrastructure. A special template provided by JATLite offers the user numerous predefined Java classes that facilitate agent constructions.

JATLite differs from traditional agent systems because it uses its unique AMR solution. Where the traditional agent systems use an *ANS* (Agent NameServer) for making connections between agents, JATLite's agents make a connection to the AMR. The AMR then forwards all agent messages by name to the last known *IP address*. Messages are buffered and saved in the AMR (like in an email system) until a receiving agent acknowledges receipt and sends a delete message to the AMR.

VI. CONCLUSION

A major drawback of agent technology is the understanding of what exactly an agent is. Missing standardization and varying definitions make it difficult to become acquaint with agents, especially when reading about basics. With help of considered examples of agent technology one can get familiar with the idea and purposes of agents.

An annoyance is that when people talk about the future of Java agents their eyes concentrate on branches as e-business and e-commerce. Agent technology should be seen on a broader-minded sense: they are also useful on other domains where they do not need to play only with money.

Nevertheless, for now at the very end, it is just like to mention that agents might be some kind of key to the future. Once Agent-Orientated Programming is on top taking over from Object-Orientated Programming, the real brightness of agent technology will be seen.

VII REFERENCES

[1] Brantschen S., Haas T.(2002.) Agents in a J2EE World. *AgentLink News*, Issue 9, Page 15-19.
[2] Chess D., Harrison C., Kershenbaum (1995) M*obile Agents: Are They a Good Idea?* IBM Research Report [ONLINE]. T.J.Watson Research Center, Yorktown Heights, New York, 1995. Reference made the 24.5.2002.
URL: http://www.research.ibm.com/massive/mobag.ps
[3] Dannegger C. Agents in real-world Applications: living markets. *AgentLink News*, Issue 9, 2002. Page 7-10.
[4] Flores-Mendez R. A. Towards a Standardization of Multi-Agent System Frameworks. Article [ONLINE]. *ACM Crossroads Student Magazine*, 1999. Association for Computer Machinery. Reference made the 23.9.2002.
URL:http://www.acm.org/crossroads/xrds54/multiagent.html
[5] Franklin S., Graesser A. *Is it an Agent, or just a Program? A taxonomy for Autonomous Agents.* Summary [ONLINE] Institute for Intelligent Systems, University of Memphis. Reference made the 12.9.2002.
URL: http://www.msci.memphis.edu/~franklin/AgentProg.html
[6] Greenwood D. Java™ Agent Services. *AgentLink News*, Issue 10, 2002. Page 7-10.
[7] JATLite description. Documentation [ONLINE] CDR, Stanford University, 1999. Reference made the 12.9.2002.
URL:http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html
[8] Jennings N. R., Sycara K., Wooldridge M. *A Roadmap of Agent Research and Development.* Summary [ONLINE] Department for Electronics and Computer Science, University of Southampton, 1998. Reference made the 25.9.2002.
URL:http://www.ecs.soton.ac.uk/~nrj/downloadfiles/jaamas98.pdf
[9] Lawton G. *Distributed agent technology comes to the Net.* Article [ONLINE] JavaWorld, 2000. Reference made the 23.9.2002.
URL:http://www.javaworld.com/javaone00/j1-00roaming_p.html
[10] Munroe S., Ashri R., Coulter-Smith E. Prospects for Agent Technology: Interviews with Industry Experts. *AgentLink News*, Issue 10, 2002. Page 3-6.
[11] Sommers B. *Agents: Not just for Bond anymore.* Article [ONLINE] JavaWorld, 1997. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents_p.html
[12] Sundsted T. *An introduction to agents.* Article [ONLINE] JavaWorld, 1998. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto_p.html
[13] Sundsted T. *Agents on the move.* Article [ONLINE] JavaWorld, 1998. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto_p.html
[14] Sundsted T. *Agents talking to agents.* Article [ONLINE] JavaWorld, 1998. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-09-1998/jw-09-howto_p.html
[15] Tveit A. *A survey of Agent-Orientated Software Engineering.* Study/paper [ONLINE] Norwegian University of Science and Technology, 2001. Reference made the 23.9.2002.
URL:
http://www.jfipa.org/publications/AgentOrientedSoftwareEngineering
[16] Venners B. *The Architecture of Aglets*. Article [ONLINE] JavaWorld, 1997. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html
[17] Venners B. *Solve real problems with aglets, a type of mobile agent.* Article [ONLINE] JavaWorld, 1997. Reference made the 12.9.2002.
URL: http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood_p.html

