Development of Mobile Agents With Aglets
(A Java Based Tool)

Mamta Yadav 1  Preeti Sethi 1  Dimple Juneja 2  Naresh Chauhan 1
1Department of Computer Engineering
YMCAUST, Sector 6, Faridabad, India
2Department of MCA
M.M.University, Mullana (Ambala), Haryana, India

ABSTRACT
Mobile agents are a type of software agent with mobility. They autonomously travel from computer to computer performing the desired task. They can be deployed in many complex scenarios of distributed applications such as mobile data computing, networking, manufacturing and scientific computing etc. Existing client server paradigm used for distributed applications has certain drawbacks such as large overhead, security problem, huge network load and data redundancy. Mobile agents eliminate the drawbacks of existing client server paradigm by reducing network load, latency and executing asynchronously. They also make the system adaptive to changes, robust and fault tolerant. Various tools have been devised for mobile agent development. Some of them are Aglets, Voyager, Grasshopper, Tryllian, JADE, Tracy, SPRINGS. This paper focuses on Aglets, one of the best tools for agent development. It is a java based tool. It makes use of various favourable agent development features of java such as platform independence, dynamic loading, secure executions and adds on some new features necessary for mobile agent development.

1. INTRODUCTION
To explain how to develop mobile agent with aglets, it is necessary to know what are mobile agents and how they are helpful. We can then understand the tool “aglets” for building agents. But before learning the tool we should also know why java is advantageous for building agent development tools.

1.1 What Are Mobile Agents?
Mobile agent is a type of software agent. It travels across a network performing tasks on systems that has agent hosting capability. It is a process that can transport its state and data from one environment to another and is capable of performing appropriately in the new environment. It itself decides when and where to move. When it moves it saves its own state and transports this saved state to the new host, and resumes execution from the saved state, It can choose to migrate between computers at any time during their execution. This property makes it a powerful tool to be used in implementation of distributed applications in a computer network. Besides mobility it also has features of autonomy, social ability and learning.

1.2 Mobile Agent Based Applications
Mobile agents can be deployed in many complex fields such as mobile data computing, electronic commerce, networking, manufacturing and scientific computing. Following are the examples of existing mobile agent based applications:
- Network management
- Grid computing and grid services
Mobile agents have also been applied in several other domains such as security for example in intrusion detection, affective computing, climate environment and weather, e-learning, semantic web services etc.

### 1.3 Why Mobile Agents?

Today, the central paradigm use to link all distributed object technologies is a synchronous message-passing paradigm where by all objects are distributed, but stationary, and interact with each other through message-passing. This paradigm is incomplete, and needs to be enhanced with additional paradigms such as asynchronous message-passing and mobility. Mobile agents can provide a single uniform paradigm for distributed object computing, encompassing synchrony and asynchrony, message-passing and object-passing, and stationary objects and mobile objects[1]. Some of the major advantages on mobile agents are given below:

- **Object-passing**: When a mobile agent moves, the whole object is passed, its code, data, execution state, and travel itinerary are all passed together.
- **Autonomous**: The mobile agent contains sufficient information for it to decide what to do, where to go, and when to go.
- **Asynchronous**: The mobile agent has its own thread of execution and can execute asynchronously.
- **Local interaction**: The mobile agent interacts with other mobile agents or stationary objects locally. It can dispatch messenger agents or use surrogate agents to facilitate remote interaction.
- **Disconnected operation**: The mobile agent can perform its tasks whether the network connection is open or closed. If the network connection is closed and it needs to move, it waits until the connection is reopened.
- **Parallel execution**: Multiple mobile agents can be dispatched to different sites to perform tasks in parallel.

The technical advantages of mobile agents are many, and there is no single alternative to all of the functionality they provide. In addition to supporting existing network services, mobile agents are also likely to make possible new services and thus new businesses.

### 1.4 Why Java For Agent Development and Introduction To “Aglets”

Some properties of java are listed below that made it better language for mobile agent programming[2].

- **Platform-independence**: Java is designed to operate in heterogeneous networks. To enable a Java application to execute anywhere on the network, the compiler generates byte code. For this code to be executed on a given computer, the Java runtime system needs to be present. Primitive data types are rigorously specified and not dependent on the underlying processor or operating system. Even libraries are platform-independent in the system.
- **Secure execution**: Java is intended for use on the Internet and intranets. It provides a secure execution. For example, Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data. It does not allow illegal type casting or any pointer arithmetic.
Programs are no longer able to access the private data in objects that they do not have access to. This prevents most activities of viruses. Even if someone tampers with the byte code, the Java runtime system ensures that the code will not be able to violate the basic semantics of Java. The security architecture of Java makes it reasonably safe to host a not trustable agent, because it cannot tamper with the host or access private information.

- **Dynamic class loading**: This mechanism allows the virtual machine to load and define classes at runtime. It provides a protective name space for each agent, thus allowing agents to execute independently and safely from each other.

- **Multithread programming**: Agents are autonomous. An agent executes independently of other agents residing within the same place and Java allows each agent to execute in its own thread of execution autonomously. Java allows multithread programming and also supports a set of synchronization primitives that are built into the language. These primitives enable agent interaction.

- **Reflection**: Java code can discover information about the fields, methods, and constructors of loaded classes, and can use reflected fields, methods, and constructors to operate on their underlying counterparts in objects, all within the security restrictions. Reflection accommodates the need for agents to be smart about themselves and other agents.

Although the Java language system is highly suitable for creating mobile agents, we should be aware of some significant shortcomings. Some of these shortcomings like inadequate resource control, no protection of references, no object ownership of references, no support for preservation and resumption of the execution state.

### 2. WORKING WITH AGLETS

#### 2.1 Aglets overview

Java and applet have revolutionized the Web. Applets are sets of program code that can be downloaded, instantiated, and executed in Web browsers. The Java Aglet API (J-AAPI) is a proposed industry standard for Java based mobile agents. J-AAPI was developed by a research team at the IBM Tokyo Research Laboratory in Japan. J-AAPI is completely written in Java, thus allowing a high portability of both the agents and the platform. The agent developed using the tool is itself called an aglet. The aglet represents an addition to applet allowing program code to be transported along with state information. The goal was to bring the flavour of mobility to the applet. An aglet that executes on one host can suddenly halt execution, be dispatched to a remote host, and resume execution there. When the aglet moves, it takes along its program code as well as its data. Java based security implementations take care of authorised access to local resources at the remote hosts. Installation of Aglets is usually composed of the following independent parts.

- **Aglet mobile Agent Platform (MAP)**: is the core platform, able to manage mobile agents.
- **Tahiti**: is the main server in charge of managing the mobility of agents. It comes with a graphical User Interface the helps administration taking care of running agents.
- **Aglets Software Development kit(Aglets SDK–ASDK)**: is a library that provides developers all the facilities required to write mobile agents complaint to the aglets.

If you want to host agents on your machine, you need to run the MAP, that acts as a daemon waiting for new agents to be created. Agents can be created locally or can be received from a remote MAP, in the latter case there is not a real “agent creation”. Since the agent already exists somewhere and is just transferred to the local MAP.
2.2 Basic Elements of Aglet Object Model

The Aglet object model (AOM) is designed to exploit the strengths of Java, including platform independence, secure execution, dynamic class loading, multithreaded programming and object serialization[3]. At the same time, Aglets provide additional capabilities for features such as resource control, protection and object ownership of references, and support for preservation and resumption of execution state. The basic elements of the AOM comprise three key abstractions.

- **Aglet**: a mobile java object that can move around aglet-enabled hosts within an environment.
- **Proxy**: a representative of an aglet that serves as a shield, protecting direct access to the public methods and providing location transparency for the aglet.
- **Context**: the aglets workplace, a stationary object that provides a uniform execution environment. Many aglets can exist in a single context, and a single computer may run multiple contexts.

The aglet package consists of a set of Java API’s and a run time environment. Each aglet is a separate class and loaded into the Aglet Tahiti server where the aglet is created and begins execution. For mobility, each machine must be running a Tahiti server. An aglet has a number of fundamental operations like creation, cloning, disposing, deactivation, activation, dispatching and retracting. A messaging framework supports inter-aglet communication. The principle method for aglet communication is through message passing. A simple event scheme means that it is only necessary to implement message handlers in an aglet for those kinds of messages that are expected. The object-based messaging framework provided is location-independent, so that two aglets need not exist within the same context or in the same location to communicate. Both synchronous and asynchronous messaging is provided, blocking operators may halt execution of an aglet until its companion replies to the message, or execution can continue through the use of a dummy object. Aglets also support multicasting. Aglets can subscribe to one or more multicast messages and listeners for those messages are then implemented to handle occurrences.

2.3 Aglets : Life cycle

Basic operations that govern the life cycle of aglets are:-:

- **Creation**: The creation of an aglet takes place in a context. The new aglet is assigned an identifier, inserted into the context, and initialized. The aglet starts executing as soon as it has been successfully initialized.
- **Cloning**: The cloning of an aglet produces an almost identical copy of the original aglet in the same context. The only differences are the assigned identifier and the fact that execution restarts in the new aglet. The execution threads are not cloned.
- **Dispatching**: Dispatching an aglet from one context to another will remove it from its current context and insert it into the destination context, where it will restart execution (execution threads do not migrate). We say that the aglet has been “pushed” to its new context. Dispatching causes an aglet to suspend its execution, serialize its internal state and byte code into the standard form and then to be transported to the destination. On the receiver side, the Java object is reconstructed according to the data received from the origin, and a new thread is assigned and executed.
- **Retraction**: The retraction of an aglet will pull (remove) it from its current context and insert it into the context from which the retraction was requested.
- **Activation and deactivation**: The deactivation of an aglet is the ability to temporarily halt its execution and store its state in secondary storage. Activation of an aglet will restore it in a context.
- **Disposal**: The disposal of an aglet will halt its current execution and remove it from its
current context.

- **Messaging**: Messaging between aglets involves sending, receiving, and handling messages synchronously as well as asynchronously.

![Aglets life cycle](image)

**Fig 1 Aglets life cycle**

### 2.4 Aglet Programming

The Aglet class is the key class in J-AAPI. It is the abstract class an aglet developer will use as base class when you create a customized aglet. The Aglet class defines methods that allows the aglet to control its own life cycle, methods for cloning, dispatching, deactivating, and disposing of itself. It also defines methods that are intended to be overridden in its subclasses by the aglet programmer, and provide the necessary facilities to customize the behaviour of the aglet. These methods are systematically invoked by the system when certain events take place in the life cycle of an aglet. Table 1 below shows the relationship between these life cycle events and methods in the aglet.

<table>
<thead>
<tr>
<th>The event</th>
<th>As the event takes place</th>
<th>After the event has taken place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>onCreation()</td>
<td></td>
</tr>
<tr>
<td>Cloning</td>
<td>onCloning()</td>
<td>onClone()</td>
</tr>
<tr>
<td>Dispatching</td>
<td>onDispatching()</td>
<td>onArrival()</td>
</tr>
<tr>
<td>Retracting</td>
<td>onReverting()</td>
<td>onArrival()</td>
</tr>
<tr>
<td>Disposal</td>
<td>onDisposing()</td>
<td></td>
</tr>
<tr>
<td>Deactivation</td>
<td>onDeactivating()</td>
<td></td>
</tr>
<tr>
<td>Activation</td>
<td></td>
<td>onActivation()</td>
</tr>
</tbody>
</table>

It is easy for you to create a customized aglet. Start by importing the aglet package, which contains all the definitions of the J-AAPI. Next, define for example the FirstAglet class, which is inherited from the Aglet class:

```java
import aglet.*;
public class FirstAglet extends Aglet { ... }
```

For instance, if you want your aglet to perform some specific initialization when it is created you just need to override its onCreation method:

```java
public void onCreation(Object init) {
    // Do some initialization here...
}
```
When an aglet has been created or arrives in a new context, it is given its own thread of execution through a system invocation of its run method. You can view this invocation as a means of giving the aglet a degree of autonomy. The run method is called every time the aglet arrives or is activated in a new context. We can say that the run method becomes the main entry point for the aglet's thread of execution. By overriding this method, you can customize your aglet's autonomous behavior.

```java
public void run() {
    // Do something here...
}
```

For example, you can use `run()` to let the aglet dispatch itself to some remote context. You can do so by letting the aglet call its dispatch method with the Uniform Resource Locator (URL) of the remote host as the argument. This URL should specify the host and domain names of the destination context, and the protocol (atp) to be used for transferring the aglet over the network:

```java
dispatch(new URL("atp://some.host.com"));
```

When `dispatch()` is called the aglet will disappear from your machine and reappear in the same state at the specified destination. First, a transfer protocol is used to bring the aglet (byte code and state information) safely over the network. Next, a special technique called object serialization is used to preserve the state information of the aglet. It makes a sequential byte representation of the aglet that later can be de-serialized to recreate the aglet's state.

Aglets can also receive messages. Message-handling in the aglet follows a callback scheme. When a message is sent to the aglet, its `handleMessage` method is invoked with the actual message passed along as argument. It is now up to this method to handle the incoming messages. It should return "true" if a given message is handled otherwise it should return "false". The sender will then know if the aglet actually handled the message.

3. APPLICATIONS OF AGLETS

1. **Data collection**: Aglets have an itinerary and can travel sequentially to many sites. One natural application of aglets is collecting information spread across many computers hooked to a network. An example of this kind of application is a network backup tool that periodically must look at every disk attached to every computer hooked to a network.

2. **Searching and filtering**: On behalf of a user, a mobile agent could visit many sites, search through the information available at each site, and build an index of links to pieces of information that match a search criterion.

3. **Monitoring**: Sometimes information is not spread out across space as on the disks of many different computers linked to the same network, but across time. New information constantly is being produced and published on the network. Aglets can be sent out to wait for certain kinds of information to become available.

4. **Targeted information dissemination**: Another potential use of aglets is to distribute interactive news or advertising to interested parties.

5. **Negotiating**: Besides searching databases and files, aglets can gain information by interacting with other agents. If, for example, we 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.

6. **Parallel processing**: Given that aglets can move from node to node and can spawn subagents, one potential use of mobile agent technology is as a way to administer a parallel processing job.

7. **Entertainment**: In this scenario, aglets represent game players. The agents compete with one another on behalf of the players. Each player would program an agent with a strategy, then send the agent to a game host.

### 4. CONCLUSION

Use of Mobile agents can revolutionize areas like mobile Data Computing, electronic commerce, networking, manufacturing and scientific computing and many more. To develop agents java is an obvious choice since it has certain traits like platform independence, secure execution, dynamic class loading, multithread programming etc which makes it good for agent development. But there are certain weakness of java like inadequate support for resource control, no protection of references, no object ownership of references, no support for preservation and resumption of the execution state. “Aglets “ the tool explained in the paper is a java based tool that provides all the goodness of java and removes the weaknesses by adding certain concepts. Aglets provide a single uniform paradigm for distributed object computing which includes providing synchrony and asynchrony, message-passing and object-passing, and stationary objects and mobile objects all together.

### REFERENCES