User Interfaces with Java Mobile Agents: The AgentSpace Case Study

Alberto Rodrigues da Silva
Universidade Técnica de Lisboa (IST)
alberto.silva@acm.org

Miguel Mira da Silva
Universidade Técnica de Lisboa (IST)
mira-da-silva@ip.pt

Artur Romão
Universidade de Évora
a.romao@computer.org

1. Introduction

The process of developing agent-based applications requires at least two tasks that are usually tackled separately by programmers. On one hand, programmers need to develop business rules and other support tasks for agents. On the other hand, programmers need to develop user interfaces (UI) for agents in order to enable end-users (not only owners but also other third parties) to interact directly with them.

This paper focuses on this second task (developing user interfaces) and describes the solutions offered by the AgentSpace mobile agent system [1,2]. Basically, we present and discuss two complementary ways to gather user-interfaces with mobile agents. On the one hand, mobile agents don't provide any UIs. This situation promotes the separation of the UI and the backend (i.e., the agent) which allows flexibility and reuse. On the other hand, mobile agents provide by default UI components, which consequently promotes agents as better units of development and management. This situation can be very suitable in the context of dynamic, large-scale applications such as those found on telecommunication and e-commerce domains.

2. Two Models of End-Users Interaction

2.1. The Conventional Model

In general, the UIs of Java mobile agents are Java applets (or related AWT components). Applets can interact with one or more agents by invoking their methods when asking for well known services, obtaining information, etc. In those cases the implementation of these UIs is done by applets. This may be a solution to develop agent-based applications: agents and applets are developed incrementally, in parallel but separately.

For example, as shown in Figure 1, the agent class `AgentA` is defined in conjunction with the applet class `AppletA` as well as an associated HTML document `HtmlA`. In order to have access to, and manage, an agent, the user accesses the `HtmlA` document, which contains `AppletA`. Then, `AppletA` obtains a reference to an instance of `AgentA`, which is used to interact with that agent.

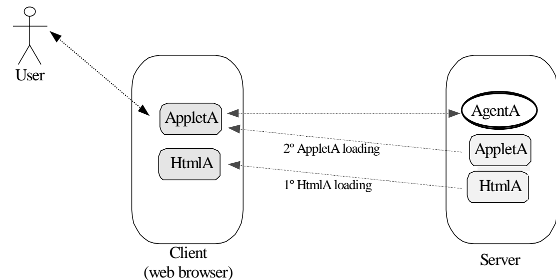

Figure 1. Conventional model for the user-agent interaction

2.2. The Integrated Model

As an alternative to the previous model, AgentSpace offers a more integrated and easy-to-use model based on the agent ability to provide its own UI componnets. In this "integrated interaction model" between agents and users, there is no need for specific applets for each agent class. It is only necessary to develop "relatively generic" applets (or stand alone applications), such as the AgentSpace-Client tool [2] depicted in Figure 2. The tool supports management and presentation of UI components defined specifically for each agent.

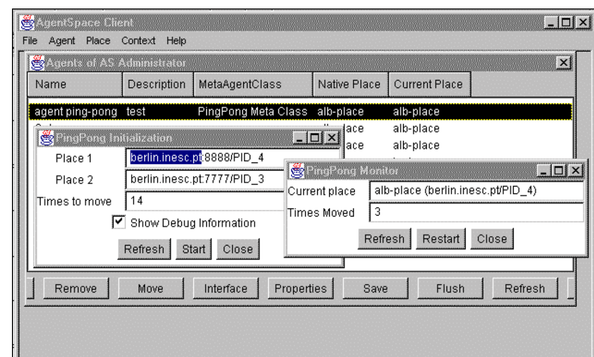

Figure 2. The AgentSpace-Client tool.

The graphical interfaces of an agent are returned in two callbacks of its respective class: `doInitialization Interface` and `doManagementInterface`. These methods are obtained through the `AgentView`'s `getInitializationInterface` and `getManagement Interface` methods, respectively. Both methods return an instance of `java.awt Frame`, that is, a Window.

The main difference between the initialization interface (`InitializationInterface`) and the management interface (`ManagementInterface`) is that the former is obtained automatically after the creation of the agent, and the latter has to be requested explicitly. Thus, the initialization interface is accessed only once, by the agent owner. On the other hand, the management interface may be obtained many times by different users.

The code below illustrates the definition of graphical interfaces (i.e., instances of `Frame`) for the specific case of the management interface.

```
public class AgentWithFrame extends Agent
{
  ...
  public Frame doManagementInterface
                  (InternalUser user) {
    Frame frame;
    if(user.equals(getOwner())
      frame= new MyPrivateFrame();
    else
      frame= new AuthenticationFrame();
    return frame;
  }
...
}
...
public class AuthenticationFrame extends Frame
{...}

public class MyPrivateFrame extends Frame {...}
```

The `doManagementInterface` method creates and returns one (out of two possible) instances of `Frame`, depending on the user that invoked the method. If it is the agent's owner, then an instance of `MyPrivateFrame` is returned. Otherwise the method returns an instance of `AuthenticationFrame`, which is defined internally in the agent's class. This interface management policy could be extended in order to integrate other criteria: for example, depending on the agent's state, user groups defined by the agent itself, etc.

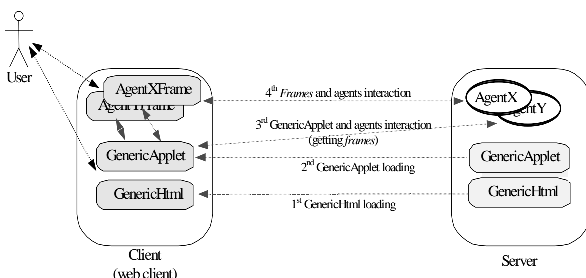

Figure 3. Integrated model of the user-agent interaction

Figure 3 illustrates the integrated model for interaction between users and agents. The main advantages of this model over the conventional model illustrated in Figure 1 are: (1) simplicity of management and maintenance (e.g., there is no need to keep multiple applet files associated to HTML documents); and (2) possibility of defining different types of interfaces to the same agent. On the other hand, this model may reduce the flexibility, reuse and innovation that can occur independently on both ends at any time. This trade-off should be conveniently analyzed for each particular application.

It is worth noting that the issues related to the interaction between agents, or between agents and other objects (in particular, `Applet` objects) are similar. The mechanism of these interactions is composed of the following main steps: (1) publishing, sharing and knowledge of agent identifiers; (2) acquisition of references to agents; and (3) invocation of agents' final methods, some of which have a higher level of variability, such as `sendMessage`, `getManagementInterface` `doOperation`, or `moveTo`.

3. Conclusions

In this paper we introduced some issues regarding user interface in the context of Java agent-based applications.

We discussed the *conventional model* of user-agent interaction, which is mainly based on HTML and Java applets, and concluded that in several situations it is not suitable and that it suffers from management, maintenance, and eventually scalability problems. In order to cope with these limitations, we proposed the *integrated model*, in which the end-user interface belonging to each agent is developed and kept together with the rest of the agent's code. It should be emphasized that AgentSpace supports both models.

Concrete agent-based applications have been developed on top of AgentSpace and are available for download from our web site [2]. The applications we built suggest that (1) the agent paradigm is suitable to design and develop many applications, namely in dynamic and distributed environments; and (2) the integrated UI model is better than the conventional model in many cases.

4. References

[1] A. Rodrigues da Silva, M. Mira da Silva, J. Delgado. AgentSpace: An Implementation of a Next-Generation Mobile Agent System. In K. Rothermel, F. Hohl (editors). *Lecture Notes in Computer Science 1477* (*Mobile Agents'98*) Springer, 1998.

[2] AgentSpace Web Site. 1997-1999. http://berlin.inesc.pt/agentspace/