Implementing a mobile agent infrastructure on the .NET framework

Antonio Boccalatte, Alberto Grosso, Christian Vecchiola
DIST – University of Genoa
Via Opera Pia 13
16142, Genova, Italy
{nino, agrosso, christian}@dist.unige.it

ABSTRACT

This paper presents the solution adopted by the AgentService platform in implementing a software infrastructure for mobile agents. The mobility service takes advantage of the agent model provided by the platform which offers the separation between the state of the agent and its activities. The modular architecture of the platform allows an elegant integration of the mobility service whose implementation resides within an additional platform module. The mobility infrastructure offers its services to the agents that can be stopped, moved, and restarted in a transparent manner. AgentService provides a sort of weak mobility service: during a transfer the state of the agent is maintained while the activities it performs are started from the beginning. The mobility infrastructure is a component that enriches the platform features and allows the implementation of more complex services such as load balancing strategies among different AgentService installations.

Keywords

Agent Mobility, Load Balancing Policy, Agent Framework

1. INTRODUCTION

Software mobility is a software property which can bring robustness, performance, scalability or expressiveness to systems [Kar98a]. Code mobility concerns the ability to migrate a unit of running code from one host to another by preserving partially or totally its execution state [Cab00a]. In particular, systems that completely maintain the execution state are said to support *strong mobility*, while systems that discard the execution state are said to provide *weak mobility*. The possibility of moving running code among computing environments is an interesting opportunity for dynamic intelligent agent systems. Agents are autonomous, pro-active, and socially able: the ability to move and to migrate between different nodes of the community enhances the previously cited features. Hence, as for objects, mobility is an interesting property for agents and mobile agents have emerged as a paradigm for structuring distributed applications.

Journal of .NET Technologies
Copyright UNION Agency – Science Press,
Plzen, Czech Republic

A definition which sufficiently characterizes the essence of a mobile agent system has been proposed by Chen and Nwana [Che95a, Nwa96a]: "*..a mobile agent is a software entity which exists in a software environment. It inherits some of the characteristics of an agent. A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a communication model and finally a navigation model.*".

The idea of mobile agent that cannot be implemented without providing multi-agent systems with a software infrastructure that allows the transfer of agents in a transparent manner. This paper describes the solution adopted within the AgentService framework to implement mobility for agents. The proposed solution turns out to be very effective thanks to the agent model adopted by the framework which separates the state and the behavior of a software agent. By using serialization and reflection the state of the agent and some information of the active behavior of the agent are persisted and transferred to the target site. In the following, the authors will illustrate the main features of the AgentService framework and will explain in detail the architecture and the implementation of the mobility infrastructure. A use case will also explain

how to use the mobility infrastructure to implement more complex services such as load balancing among different AgentService installations.

2. AGENTSERVICE

AgentService Main Features

The agent oriented paradigm [Wol99a] can be a useful abstraction to model open and dynamic communities. An agent is an autonomous software entity provided with some levels of "intelligence". Moreover, by means social ability, agents can enhance their performances by interoperating in communities called multi-agent systems (MASs) [Wei99a]. A widely accepted architecture specification for multi-agent systems with a reference agent model is the one proposed by the Foundation of Intelligent Physical Agents (FIPA) [Fip01a].

AgentService [Boc04a] is a framework for developing multi-agent systems based on the Common Language Infrastructure, whereof .NET framework is one implementation. AgentService provides a specific agent model and a runtime environment for agent execution compliant with the FIPA specifications. In literature there are many works concerning agent platforms, the most interesting and known are Zeus [Nwa98a], FIPA-OS [Pos00a], and JADE [Bel99a]; AgentService is characterized by an extremely modular architecture and a flexible agent model which allows the implementation of different agent architectures.

Two different kinds of modules have been designed to model all the features of the AgentService platform: core modules and additional modules. Core modules implement all the services required by the platform instance to set up its activity; they involve management of assemblies in which agent templates are defined (Storage Module), the messaging service, the management of the agent's persistence, and a logging service. Through additional modules new features can be added to the platform: they enrich the platform capabilities but they are not essential for the standard activity of agents.

Within AgentService, agents are designed as software entities whose activity is defined by a particular managed set of data (Knowledge objects) and performed by a set of concurrent behaviors (Behavior objects). A Knowledge object represents a set of correlated data modeling a structured concept of the problem domain. A collection of specified Knowledge objects define the state of a software agent: it can be persisted and portions of it can be shared among the different Behavior objects. Behavior objects contain all the agent computational logic and define the agent aggregate behavior. Behavior objects are concurrent and share the information they need by means of the Knowledge objects. Such distinction between activities and data allows a clear decomposition of the agent definition, represents a flexible and generic model. From the implementation point of view every agent instance is deployed in a dedicated Application Domain, which ensures the autonomy and safety of the code executed inside it (agent activities).

3. MOBILITY IN AGENTSERVICE

Introduction

According to the definition given by Chen and Nwana [Che95a, Nwa96a] a mobile agent inherits all the properties of a software agent and, in addition, contains a navigation model which embraces all aspects of agent mobility from the discovery and resolution [Whi95a] of destination hosts to the manner in which a mobile agent is transported. Hence, the introduction of a navigation model implies the extensions of the agent model defined within AgentService with the previously discussed features.

It can be observed that the ability of identifying and discovering destination hosts is already implemented by using the directory services of the platform: AgentService platforms can join together and define a federation. A federation defines the boundaries into which the mobility service takes place. In the following the authors will focus the attention on the second element characterizing the navigation model that is the machinery required to transfer agents. In order to move an agent the model defining its life cycle needs to be extended with an additional state which characterize the agent while is being moved. As suggested by the FIPA specifications the common life cycle of an agent has been extended by adding the *transit* state and two actions to enter and leave that state (*move* and *execute*). The agent itself can require the move action while the platform, through the Agent Management System (AMS), is responsible of completing the migration by performing the execute operation.

In addition, mobile agents require a suitable runtime environment which provides a transfer service allowing them to move from one node to another: this environment is built on top of a host system. Within AgentService, this runtime provides to the agent with a transfer service based on a variation of weak mobility: even if the execution does not continue exactly by executing the next instruction a partial resume of the execution state has been implemented. The main idea is to exploit the adopted agent model and move just the agent state (knowledge objects) among platforms. Within the target platform the agent activities can be restarted

taking advantage of the persisted agent state. In addition, the framework provides developers with an entry point, the *Resume* method, for checking the state and the activities of the agent before it continues the execution.

Architecture of the Mobility Service

The architecture of the mobility service takes full advantage from the agent model adopted by the platform: the separation among the agent state from the activities it performs makes the migration process simple. In order to run an agent the runtime environment needs the information defining the agent state and the assemblies containing the agent definition. Hence, moving an agent among AgentService installations requires moving its state and ensuring the presence of that agent type definition on the target platform. Once an agent is moved it is possible to restart its activity by instantiating a new agent of that specific type and restoring its state. State restoration involves loading the transferred knowledge objects and the activation of all the behaviors objects running when the agent was stopped. The information about knowledge objects and the state of each behavior (ready, active, suspended) are all what is really needed to move an agent.

The process which transfers an agent is activated by a request and can be described as follows:

1. *negotiate*: the AMS of the source target contacts the AMS of the target platform and asks if the agent can be moved;
2. *stop and persist*: if the agent can be moved, the AMS stops its activity, persists its state, and puts it into the transit state (*move* action);
3. *transfer*: the AMS instruct the mobility module to move the agent. The state of the agent is transferred to the target platform. This operation may require the transfer of the assemblies describing the types of the agent or used by it;
4. *restore*: the mobility module notifies the AMS that the transfer is completed. The AMS creates an instance of the same type of the agent received sets its state and invokes the *Resume* method allowing the programmer to customize the re-activation of the agent;
5. *execute*: the agent changes its state from transit into its original state, the AMS of the source platform is notified of the successful transfer and the agent is activated (execute action).

This process is implementation independent and AgentService defines an interface (*IMobilityModule*) that every module that wants to offer this service must implement. In this way, developers can implement the service as they prefer: a web service, an ftp service, or a custom channel.

The model adopted by AgentService to define an agent greatly simplifies the work of the mobility module. Thanks to the clear separation among the state and the activities the maintenance of the execution state is obtained by saving all the knowledge objects composing the knowledge base of the agent and the status of the behavior objects. When the agent is restored the information saved are loaded into the new instance and all the activities previously stopped are started. In particular, the mobility infrastructure has to deal with the transfer of assemblies if the repositories of the two platforms are not synchronized.

The entire process that allows mobility of agents takes place if and only if the platforms are allowed to transfer agents; otherwise it stops the *negotiation* phase. The AMS of the source platform will look in the platform configuration to determine if it is allowed exporting agents, while the AMS of the target platform will check if it is allowed to import agents. These information are contained in the platform profile and administrators can customize the platform behavior by modifying the profile in the configuration file.

4. LOAD BALANCING POLICY

Load balancing in AgentService is managed by the Load Balancing Policy (LBP) module. It provides a service that federates platform instances and creates a unique environment in which agents can move. By default, LBP comes with two policies. The first policy balances the number of agent among platforms, while the second is based on the number of exchanged messages moving in the same platform the agents interacting more frequently. The *platform context*, provided by AgentService to each module, gives access to these information. LBP modules, installed on different nodes, cooperate to constitute a federation of platforms defining the border within the balancing policies can be applied. The federation system adopts a client/server model: each node provides its platform profile to the master node which maintains the federation and applies policies. The LBP configuration file defines the structure of the federation indicating which node works as server. At runtime new platforms can dynamically join the federation by registering their profiles to the master node. The master node applies balancing policies every time an interesting event occurs (i.e. the creation of a new agent, the registration of a new

platform with the federation). Developers can implement new load balancing policies and dynamically load them in the server LBP module as plug-ins. Hence only the LBP module of the master node handles the balancing policy.

5. CONCLUSIONS

The AgentService modular architecture allows the design and the implementation of additional services which are fully integrated with the standard ones. The mobility module provides a weak mobility service even if it automatically maintains the agent state. Developers are provided with facilities for customizing agent resumption. The presence of a software mobility infrastructure allows the platform to be enriched with more capabilities. The load balancing policy (LBP) module provides an interesting service for resources management exploiting the mobility service. The LBP module allows administrators to apply load balancing algorithms to a federation of AgentService platforms. The agent transfer process was tested applying the default balancing policies. The tests pointed out that the most onerous operation during the mobility process is the transfer of assemblies required for agent activities. This is a minor drawback since in common balancing scenarios all the nodes are likely to run the same agent types; hence there is no need for huge assembly transfers.

The mobility infrastructure suffers from the lack of interoperability with different FIPA compliant platforms, in particular Jade. The development of an interoperable mobility infrastructure is a very challenging task which involves a lot of problems due to the adoption of different technologies, architectures, and agent models.

6. REFERENCES

[Kar98a] Karnik, N. M., and Tripathi, A. R. Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency 6(3), pp. 52-61, July-September 1998.

[Cab00a] Cabri, G., Leopardi, L., and Zambonelli, F. Weak and Strong Mobility in Mobile Agent Applications. Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK), April 2000.

[Che95a] Chess, D., Harrison, C., and Kershenbaum. A. Mobile Agents: Are they a good idea?. Technical Report, IBM T.J. Watson Research Center, NY, March 1995.

[Nwa96a] Nwana, H. Software agents: An Overview, Knowledge and Engineering Review 11(3), November 1996.

[Wol99a] Wooldridge, M. Intelligent Agents. In Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, pp. 27-78, 1999.

[Wei99a] Weiss, G. Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, 1999.

[Boc04] Boccalatte, A., Gozzi, A., Grosso, A., and Vecchiola, C. AgentService. The Sixteenth International Conference on Software Engineering and Knowledge Engeneering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 June 2004.

[FIP01a] FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/

[Nwa98a] Nwana, H.S., Ndumu, D.T., and Lee, L.C. ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. Proceedings of PAAM98, pp. 377-391, London, U.K., 1998.

[Pos00a] Poslad, S., Buckle, P., and Hadingham, R. The FIPA-OS agent platform: Open Source for Open Standards, PAAM2000, Machestor, UK, April 2000.

[Bel99a] Bellifemine, F., Rimassa, G., Poggi, A. JADE - A FIPA-compliant Agent Framework. Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.

[Whi95a] White J.: The foundation of the electronic market place. General Magic white paper 1995.