MAPNET: A .NET-Based Mobile-Agent Platform

Dilyana Staneva, Denitsa Dobreva

Abstract: *Mobile agent-based computing is an attractive model for organizing distributed computations. This paper reveals our experience in developing a specific implementation of this model: the MAPNET mobile-agent platform, based on the .NET Framework. The paper describes the basic functionality of the platform and focuses on the design and implementation of a MAPNET agent and core agent services. Brief directions when and how to use the platform are also given.*

Keywords: *distributed computing, mobile agent-based computing, .NET Framework*

INTRODUCTION

Mobile agent-based computing has been quite popular in academia and industry over the last 10 years, as a replacement or extension to the traditional client/server model. The majority of mobile-agent platforms currently used are Java-based. The MAPNET platform we've built is an alternative implementation of the mobile-agent model, based on the .NET Framework and the MASIF specification for mobile-agent systems behavior and interaction. The MAPNET project is generally targeting .NET developers of distributed applications. We intend to include it as case study in distributed programming courses.

BACKGROUND

In the context of the mobile-agent model, an application is organized as a collection of mobile and stationary agents. A mobile agent encapsulates code and data transfer in a networked environment and can be used as a structuring block for distributed computations.

The basic difference between stationary and mobile-agent solutions is illustrated in Fig.1.

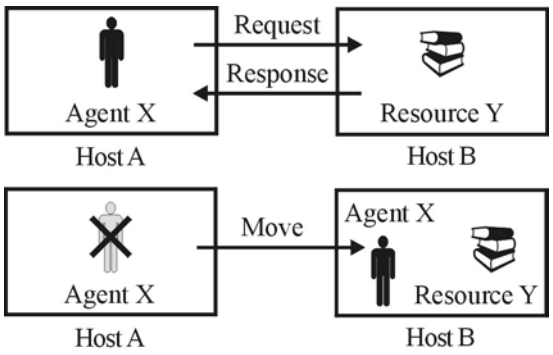

Fig.1 Stationary vs. mobile-agent solution

Instead of communicating remotely, the mobile agent migrates to or near the location of the resources needed. Replacing remote communication with migration and local communication can be advantageous in several scenarios: intensive processing, filtering or monitoring at the data source.

Mobile-agent platforms are the specific implementations of the general model, supporting a single or multiple languages for developing mobile-agent solutions. Mobility is achieved by extending existing, typically interpreted, languages. A key component of a platform is the agent server, also called agent system or agency, which hosts and manages mobile agents. The majority of mobile-agent platforms are Java-based (IBM Aglets [1], IKV++ Grasshopper [3]), while others support scripting languages (D'Agents [2], Tacoma [9]). We've applied the Grasshopper and D'Agents platforms for building

monitoring applications [7, 8]. An attractive feature of using mobile-agent platforms, especially for teaching purposes, is their high-level style of expressing migration, hiding the underlying communication details.

Distributed solutions with mobile agents are typically suitable for scenarios, in which migration could reduce network traffic [4]. Furthermore, mobile agents can be used as "pluggable" services.

As mobile-agent platforms are quite different in architecture and implementation, the MASIF (Mobile Agent System Interoperability Facilities) specification [5] is intended to facilitate their interoperability, especially for platforms using the same instrumental language. MASIF covers several aspects of agents and agent servers, including agent transfer and management, naming and security. Agents' serialization, deserialization and execution are outside the scope of MASIF. MASIF defines two public interfaces of a mobile-agent platform: the basic functionality of an agent server and of a registration system. We've basically conformed to the requirements of the MASIF specification. Several concepts and functions are not implemented in the current, initial version of the MAPNET platform; they can be added in subsequent versions.

The MAPNET platform is an implementation of the mobile-agent model, based on the Microsoft .NET Framework [6]. We've developed the platform as an alternative tool, specifically targeting .NET developers. The MAPNET project is intended to encourage students to explore diverse models of distributed computations, leveraging .NET Framework resources. The .NET Framework provides the developer with a wealth of technologies for implementing distributed solutions, like Remoting, Web Services, serialization, reflection and security.

The MAPNET platform is a type library, written in C#, for representing agents and agent servers (Fig.2), targeting the .NET CLR (Common Language Runtime).

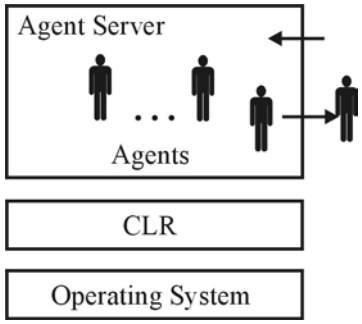

Fig.2 MAPNET components

MAPNET AGENT

A mobile agent in the general model is mapped to a descendent of the abstract base `Agent` class in the MAPNET platform. It provides the core functionality of a mobile agent, as illustrated in Fig.3.

The `Agent` class is marked as `Serializable` to enable the migration of its descendents' instances. For agents to be accessible remotely, the base `Agent` class inherits the .NET `MarshalByRef` class. The MAPNET platform supports both migration and remote inter-agent communication. It's the developer who decides whether and when to use migration or remote communication.

Migration is initiated by the agent itself through invoking its `Move()` method, passing as parameter the location of the desired destination agent server. The `Remove()` method initiates the agent's own destruction. Both methods cannot be overridden by the agent developer, as they are crucial for the functionality of a mobile agent. They call the

respective agent server methods. The agent server provides the interface between the mobile agents and the .NET Platform. Thus, each agent stores internally a reference to its hosting server.

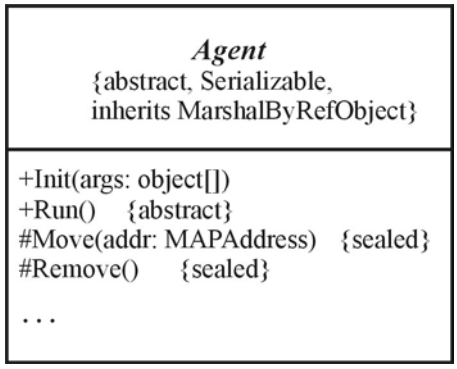

Fig.3 The base Agent class

The `Run()` method is the only abstract method of the `Agent` class. The agent developer must extend the `Agent` class and provide an implementation of the `Run()` method. The `Run()` method is executed as the "body" of the mobile agent. In it, the developer should specify the agent's behavior during its "lifetime". Here is an example how to extend the base `Agent` class:

```
[Serializable]
public class MyAgent : Agent {
   public override void Run() {
      // agent's behavior:
      // local processing first,
      // then attempt to migrate
      try {
         Move(newLocation);
      }
      catch (Exception e) {
         // migration failed
      }
   }
   ...
}
```

The `Move()` method can raise exceptions, like `AgentMigrationFailed-Exception`. Therefore, its invocation should be placed in a `try`-block with proper `catch`-filters.

There are several `Agent` methods, which the developer can optionally override. One of them is the `Init()` method, which is used in MAPNET in place of a standard constructor. It is called by the agent server after creating the agent instance. `BeforeMove()` and `AfterMove()` are respectively called immediately before and after migration. `BeforeRemove()` is called before agent destruction. The developer should override `BeforeMove()` and `BeforeRemove()` to dispose resources consumed by the agent. The `Log()` method should be implemented to write diagnostic agent-related messages to a user-specified file.

Apart from the methods, inherited by `Agent`, the programmer-defined `Agent` descendent could contain other members to define its specific state and behavior.

Each mobile agent in the MAPNET platform is assigned a profile, managed by the agent server. The profile is represented by the MAPNET `AgentProfile` class and

contains useful information: agent name, agent state, names of the home, last visited and current agent server, optional agent description, as well as programmer-defined agent properties, like "friendly" name. The mobile agent "carries" its profile with it during migration.

According to the MASIF specification, agent names, as well as agent server names, must be globally unique and immutable, but how to achieve uniqueness is implementation-dependent. Names in MAPNET are constructed in the following format:

Prefix#IPAddress:Port#DateTime,

where **Prefix** can be "Agent" or "AgentSystem". A similar approach is used in the Grasshopper platform [3].

MAPNET AGENT SERVER

MAPNET agents are executed in the context of an agent server. One or more instances of the MAPNET server must be running on each machine willing to host MAPNET agents.

The MAPNET server provides the core infrastructure for managing agents. Its main functions are grouped together as services (Fig.4).

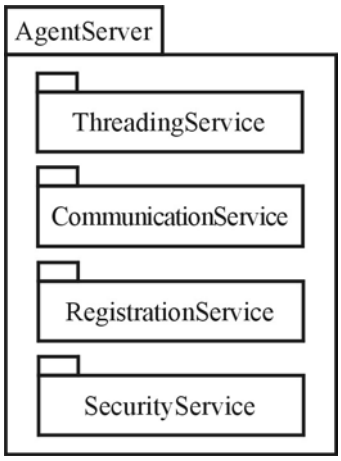

Fig.4 MAPNET core services

The main functions of the MAPNET server can be summarized as follows:

- Agent management: According to MASIF, each agent server must be able to create, terminate, suspend, resume and transfer (send and receive) mobile agents,
- Agent naming,
- Threading: The thread manager assigns a separate background thread to each newly-created or newly-arrived agent to carry out its actions,
- Agent registration: The local registration service within each server is used to store and retrieve agent profiles. New agents are automatically registered. Provides methods for searching local agents,
- Remote communication between agents, and
- Security: The server must provide a secure environment for agents.

The MAPNET communication service is responsible for all interactions between agent servers. It implements the infrastructure for migration and remote inter-agent communication, based on the .NET Remoting technology. The currently implemented security mechanisms: agent server authentication and message encryption, are embedded into the communication service.

A supporting class-loading service is used to dynamically load the necessary mobile-agent related assemblies and classes.

The MAPNET server is responsible for starting and stopping all services, and forwarding invocations of its interface methods to the respective services.

The server exposes a set of agent-related events, like `AgentCreatedEvent`, `AgentRemovedEvent`, and `AgentMovedEvent`, to notify interested "listeners". Event publishing and subscription is a very flexible pattern of interaction and synchronization, which we've applied in our implementation. For example, the GUI front-end to the MAPNET server subscribes and reacts to agent-related events.

Each agent server is associated with a profile, containing its name, optional description and properties.

The thread manager within the MAPNET server organizes a pool of threads and assigns them to mobile agents. We've developed a custom pool to minimize thread creation overheads. The implementation of the pool relies on .NET `WaitHandle` objects. The .NET Framework provides a `ThreadPool` class, but its threads are not controllable, e.g. suspendable or resumable.

During its lifetime, an agent can be in one of the states, depicted in Fig.5.

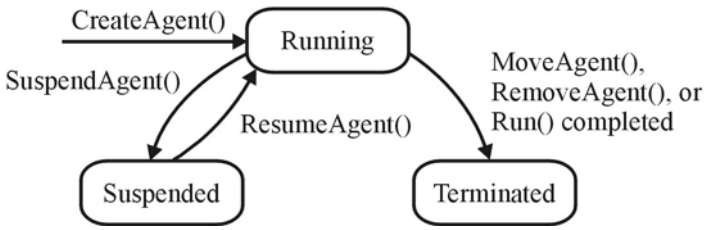

Fig.5 Mobile agent states

`CreateAgent()`, `SuspendAgent()`, `ResumeAgent()`, `MoveAgent()` and `RemoveAgent()` are MAPNET server methods, implementing the respective control operations on agents. An agent is destroyed and its thread freed, when its `Run()` method completes or when the agent initiates its own destruction (`Remove()`). Agent migration is implemented, following the requirements of MASIF, in several steps. At the source agent server, migration includes three main actions:

- The agent is terminated and its controlling thread is returned to the pool, ready to serve other agents.
- The current agent's state is converted to a byte stream. Our current implementation relies on the .NET binary serialization. Therefore, the agent's class must be marked as `Serializable`. Binary serialization affects all fields, not explicitly tagged with the `NonSerialized` attribute.
- The actual agent transfer is delegated to the communication service.

The destination agent server performs a symmetric sequence of steps:

- Its communication service receives the agent.
- The agent's state is restored through deserialization.
- The agent is assigned a controlling thread to execute its `Run()` method.

CONCLUSIONS AND FUTURE WORK

During the design and implementation of the MAPNET mobile-agent platform, we've tried to follow the requirements of the MASIF specification and leverage the most suitable .NET Framework resources to build a functional and elegant product.

We've relied heavily on .NET threading, remoting, serialization, event handling and security.

We've also implemented a global service for registering agents and agent servers, based on .NET Remoting. The current, initial implementation does not support MASIF places and regions.

Two immediate directions for future work will be oriented to strengthening security by enforcing agents' permissions, and extending the class-loading service.

The MAPNET platform could be useful for developers working on related problems, as well as for those intending to use a .NET-based mobile-agent platform. We consider using the project as case study in distributed programming courses.

REFERENCES

[1] Aglets Specification 1.1. IBM Corp., 1998. http://www.trl.ibm.co.jp/aglets/.

[2] D'Agents Platform. http://www.cs.dartmouth.edu/~agent.

[3] Grasshopper Programmer's Guide. IKV++ GmBH, Informations- und Kommunikationssysteme, 2001. http://www.grasshopper.de.

[4] R. Gray, D. Kotz, G. Cybenko and D. Rus. Mobile agents: Motivations and state-of-the-art systems. Dept. of Computer Science, Dartmouth College, 2000.

[5] MASIF Specification. http://www.omg.org, 1998.

[6] Microsoft .NET Framework. http://msdn.microsoft.com/netframework.

[7] D. Staneva, P. Gacheva. Building distributed applications with Java mobile agents. Proceedings of Next Generation Network Technologies International Workshop, pp. 103-109, Rousse, Bulgaria, 2002.

[8] D. Staneva, E. Atanasov. Using Tcl Mobile Agents for Monitoring Distributed Computations. Proceedings of the International Conference on Computer Systems and Technologies CompSysTech'2003, pp. II.21.1-II.21.6, Sofia, Bulgaria, 2003.

[9] Tacoma Mobile-Agent System. http://www.tacoma.cs.uit.no.

ABOUT THE AUTHORS

Dilyana Violinova Staneva, Assistant Professor, Ph.D. Computer Science & Engineering Department, Technical University of Varna, Bulgaria. Phone: +359 52 383424, e-mail: dilyana@tu-varna.acad.bg.

Denitsa Yordanova Dobreva, Worked on the MAPNET platform as a master student at the Computer Science & Engineering Department, Technical University of Varna, Bulgaria.