Design and Implementation of the Mobile Agent Platform Crossbow

Mario Kusek, Dubravko Voncina and Vedran Vyroubal
Department of Telecommunications
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
Email: mario.kusek@fer.hr, dubravko.voncina@zg.hinet.hr, vedran.vyroubal@inet.hr

Abstract—Basic requirements for the new generation networks are applications that offer full mobility of users and services. Agent concepts and mobile software agents have become a part of the system and service architecture of the new generation networks. This paper presents mobile agent platform Crossbow. Crossbow architecture, design and implementation are explained and compared to other platforms.

I. INTRODUCTION

New generation networks (NGN) require applications that offer full mobility of users and services. Agent concepts and mobile software agents have become a part of the system and service architecture of the new generation networks.

Mobile agent platform Crossbow was developed as the core platform, which can be used for researching complex mechanisms such as agent communication, cooperation, security, etc. Fundamental requirements were: to achieve basic functionality (agent creation, migration, etc.), good performance, ease of use and execution in heterogeneous environment. All four requirements do not support each other. For example, ease of use is in contradiction with good performance. Java was selected for the implementation because programs written in Java can be executed in a heterogeneous environment.

II. ARCHITECTURE

Agent platform is the infrastructure, which provides the environment that makes agent execution possible. Agent is executed within that environment and cannot “live” outside of it. Each node in the network, at which it can be executed, must have at least one agent system running. Agent system, also known as agent server, is a server software that provides agent environment. An agent platform implementation must provide:

- agent management – creating, starting, migrating agents
- agent communication
- supervision of agents, error notification mechanisms
- security mechanisms

In order to communicate with another agent, the agent that initiates communication must know the address of the destination agent. Mechanisms for providing information about agents that are present in the system must be included in the agent system as well as mechanism for agent identification. Each agent in the system must have a unique name that identifies that agent. An agent in Crossbow is identified by its unique name which consists of: the name of the host where the agent was created (home agent system), TCP port at which agent server is running and the sequential number of that agent. Besides agent identification, secure communication and secure migration of agents also have to be provided by the agent system.

Components of the agent platform can be divided into three layers depending on their functionality (Figure 1):

Agent Layer: is responsible for agent management i.e. agent creation, preparing agent for migration and agent execution.

Security Layer: enables usage of secure protocols and data encryption, which also disable unauthorized access to different resources of the agent system.

Communication Layer: handles agent communication (sending and receiving messages), communication protocols and data formats. It performs tasks such as object serialization and message delivering, communication between agent systems, etc.

Figure 2 shows the basic structure of the Crossbow agent platform. Dotted arrows represent communication that might
or may not be realized (different requests require different actions) while lined arrows represent communication between objects that is always performed.

AgentServer, IndividualServer, MessagesRepository and MessagesFolder are the most important parts of the agent layer. Other components, as well as IndividualServer, have the functionality of the communication layer. Components of the security layer are not yet developed, but it is presumed that IndividualServer will also be a significant part of this layer.

Every agent system has exactly one MessagesRepository and one LocationsRepository object.

MessagesRepository: contains messages for all agents that are currently settled on agent platform.

MessagesFolder: stores messages for an agent. Every agent has its own MessagesFolder.

LocationsRepository: contains information about current location of all agents that were created on that agent platform and were not yet destroyed.

Server: Its function is to listen at the specified TCP port. After receiving request for communication, Server starts a new thread that contains one IndividualServer object.

IndividualServer: receives data from the network and analyses it. Depending on the type of received data IndividualServer exchanges information with other objects such as MessagesRepository or LocationsRepository.

CrossbowClassLoader: gets appropriate agents class from class repository after IndividualServer received data with an agent object or request for creation of a new agent.

AgentServer: is the interface between the agent and other components of the agent system. It retrieves messages from MessagesRepository, analyses and adjusts message data and, finally, invokes one of the agent’s methods with arguments that were contained within the message. AgentServer also has important role in processes of agent migration and agent destruction. Every agent has its own AgentServer. Looking from the perspective of the agent system, every AgentServer contains only one agent. AgentServer also supervises and controls the agent serialization process.

AgentProxy: is not visible in the figure 2 but objects of this type are used as interfaces between user and agents. Normally, at least one AgentProxy should be created for each agent. By using AgentProxy, the user can send a message to the remote agent without knowing the exact location of that agent.

III. BASIC MECHANISMS

A. Agents’ Structure

Two types of agents exist in Crossbow: mobile and stationary. Agent must be instantiated from the class that extends either the Agent class, for mobile agents, or StaticAgent, for stationary agents. Each agent can be in active or passive state. While being in passive state, agent awaits for incoming events and does not execute anything. Transition from passive to active state occurs when the agent server receives a message and delivers it to the passive agent.
Agents exchange messages over the MessagesRepository object. When an agent wants to send a message to another agent it sends it to MessagesRepository which translates it to destination agent.

A stationary agent executes its job on the agent system it was created and can not be migrated to another agent system. That is because the StationaryAgent class does not implement the Serializable interface. The Agent class implements that interface which enables a mobile agent to be serialized to the stream of bytes and migrated to another agent system over an TCP/IP network. However, that does not mean that a stationary agent can not be created on a remote agent system. Stationary agents can provide a single generic way for other agents to interact with non-agent software systems [1].

B. Creating Agents

An agent can be created on a local or remote agent system. When an agent is created on a remote agent system, all the data that is needed for agent initialization must be serializable. Otherwise the agent will not be created and an exception will occure.

When a user initiates agent creation it calls the createNewAgent method in the Server class with parameters: the class name of the agent and the address of the following destination agent system (figure 3). After that, Server serializes parameters and sends them to remote Server. Remote Server creates a new IndividualServer which loads agent’s class, creates AgentLocation in the LocationsRepository, creates MessagesFolder in MessagesRepository and finally creates AgentServer with the agent within. After agent creation remote Server returns the agent ID to the Server which creates AgentProxy that points to the created agent. Finally, AgentProxy object is returned to the user.

C. Finding Agents

AgentProxy is a communication interface between the user and the agent. It simplifies communication and finds the current location of an agent. Agent’s location data is saved in LocationsRepository at the agent system where the agent was created. AgentProxy connects to that agent system and requests agent location for the agent with specified ID. Server object in the agent’s agent system creates a new IndividualServer which contacts LocationsRepository. LocationsRepository returns the agent’s location data to the IndividualServer which sends it back to the AgentProxy.

D. Agent Migration

An agent can migrate autonomously or on demand. Both ways require an appropriate migration message to be put into
Fig. 4. Agent Migration

E. Agent Removal

This agent platform does not have distributed garbage collection which would automatically remove unreferenced and inactive agents from the system. That is because of the complexity of such a mechanism. Manual agent removal can be started in two ways: by calling the destroyAgent method inside the agent or by calling the destroyAgent method on the AgentProxy. Destruction process starts when agent’s AgentServer object receives a destruction message. After that it clears the reference to the agent and deletes the MessagesFolder at the current location and AgentLocation at the home agent system.

IV. AGENT COMMUNICATION

Considering that mobile agents migrate between agent systems in the network, communication protocols must ensure location transparency, reliability, efficiency, asynchrony and adaptability [2]. Since mobile agents migrate autonomously in the network, they can not reliably “know” the location of their connection peer. Therefore, some kind of tracking mechanism must be built into the agent system to keep track of agent locations. This mechanism allows each entity in the system to send messages to its peer without knowing where they physically reside.

A. Location Tracking

Location tracking in Crossbow agent platform is similar to the mobile IP mechanism. This mechanism is proposed by Object Management Group in its Mobile Agent System Interoperability Facility (MASIF) [3]. The home agent system is responsible for tracking the current agent location. When the agent migrates to the new location, the home agent system is informed about the new location of that agent.

The advantage of this approach is that the location of each agent is available at all times, and retrieving the agent is very easy. The drawback is overhead in case when agent migrates often without having to receive any messages. In that case overhead is large.
B. Mailbox Model

In this model each mobile agent has a mailbox that buffers the messages sent to it. The mobile agent migrates together with its mailbox. The mobile agent itself determines when to read new messages from the mailbox and that way message receiving does not have to be synchronized with the entities that are sending messages. The communication is one way communication, from sender to receiver. If the agent that received the message wants to return the result, it has to reply with another one way message.

C. Message Passing

Since two way communication is not possible, if a user is not an agent, the user and the agent can communicate only in direction from the user to the agent. Because of that, since an agent can start the user interface, agent to agent communication is preferred.

When a user sends a message to an agent it calls either high level message passing methods or the sendMessage method in AgentProxy. High level message passing methods are: destroyAgent, invokeMethod and moveAgentTo. destroyAgent sends a destroy message to the agent. moveAgentTo sends a message that initiates agent migration. invokeMethod sends a message that invokes specified method in the agent. Parameters for the invokeMethod method are: the name of the method that should be invoked and arguments for method invocation. Arguments are packed in Vector object. By calling the sendMessage method low level message passing is started. The parameter of this method is Message object. For constructing this object, knowledge of the internal protocol is needed. This protocol could be changed in the future and that is the reason why low level message passing is not recommended for normal use and should rather be used for debugging purposes.

When the user calls the invokeMethod method, the AgentProxy creates a Message object and calls sendMessage (figure 5). The method sendMessage finds the location of the agent and sends the serialized Message to the remote agent system where the agent resides. At the remote agent system a Server object accepts the connection and creates a new IndividualServer that receives the Message object and puts it into the MessageRepository in the MessagesFolder of the specified agent. After that the agent decides when it will read that message and act upon it. Agent to agent communication, where agents are on different agent systems, is the same as the user to agent communication. When both agents reside on the same agent system, message passing between them is executed by putting message directly to another agent’s MessagesFolder. This is possible because both agents share the same MessageRepository.

V. COMPARISON TO OTHER PLATFORMS

There are lots of agent platforms today. Most popular agent platforms that are programmed in Java are Objectspace Voyager [4] and IKV++ GmbH Grasshopper [5].

A. Functional Comparison

Voyager is an ORB (Object Request Broker) powered by agents. Agents are objects that extend the Agent class. That concept is also built into Crossbow. Voyager has integrated basic security by Java security manager. Other security mechanisms like secure communication between agent system nodes and authentication by digital signatures are not implemented. It has advanced communication mechanisms like asynchronous communication, synchronous communication and group communication. Asynchronous and synchronous communication can both be one way or two way. Voyager does not support
agent standards like FIPA (Foundation for Intelligent Physical Agents) standards [6] or MASIF [3]. Voyager can not be executed on small devices like PDAs. Objectspace does not intend to upgrade agent part of Voyager with any new features.

Grasshopper is a mobile agent platform designed for the NGN (New Generation Network). Grasshopper has advanced security mechanisms which include Java security manager, agent and code authentication with X.509 certificates. SSL (Secure Socket Layer) is used for network communication. It has good communication mechanism like synchronous and asynchronous communication in two directions but does not support communication to a group of agents. Regarding standards, Grasshopper has an add-on for FIPA ACL (Agent Communication Language) and for the MASIF standard. Is has versions for execution on small devices like PDAs which can execute Java virtual machine — Personal Java.

Crossbow is designed for fast agent migration and small memory and processor consumption. This agent platform is in development process and for now it has only basics mechanisms implemented. Only Java security manager as security mechanism and asynchronous one way communication are implemented. Crossbow does not support any standard and Agent execution on small devices is not yet possible.

From table I it is visible that Grasshopper has the most advanced security mechanisms, implementation of agent standards and support for small devices while Voyager has advanced communication mechanisms. At this stage of development, Crossbow is still inferior to other two systems.

B. Speed Comparison

Platform speed measurement were executed in a laboratory with LAN network. Computers used were a PC Intel Pentium III 600MHz with Windows 2000 operating system, Java J2SDK v1.4.1 and all three agent platforms. For the purpose of the experiment an agent with simple functionality was designed. The agent was created and filled with data. After that, the agent was sent to the second computer where it wrote data into a file. Then, the agent migrated to the third computer and wrote the same data on it. Finally, the agent returned to the home computer where it made calculation and reported the time consumed for the whole operation. The experiment was repeated with different amount of data (0 bytes, 10Kb, 100Kb and 1Mb).

VI. CONCLUSION

Basic requirements for the new generation networks are applications that offer full mobility of users and services. Agent concepts and mobile software agents have become a part of the system and service architecture of the new generation networks.

In this article the design and implementation of the agent platform Crossbow were introduced. All parts of the architecture are explained in detail. Basic mechanisms such as agent creation, agent location retrieval, agent migration and agent removal were explained. Agent communication mechanisms were compared with other agent platforms. Functional comparison was elaborated and speed comparison was performed in the experiment. By the functional richness, Crossbow is inferior to other platforms, but that is also the reason why its agents are fast.

The areas for the future platform improvement are security, advanced communication mechanisms, conforming to agent standards and platform adaption to small devices. Since some of those improvements will probably degrade performance of the agent platform, they should rather be implemented as optional add-ons and not as standard features.

REFERENCES