Mob: A Scripting Language for Mobile Agents Based on a Process Calculus

Hervé Paulino[1], Luís Lopes[2], and Fernando Silva[2]

[1] Department of Informatics. Faculty of Sciences and Technology.
New University of Lisbon. 2825-114 Monte de Caparica, Portugal.
herve@di.fct.unl.pt
[2] Department of Computer Science. University of Oporto.
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal.
{lblopes, fds}@ncc.up.pt

Abstract. Mobile agents are the latest software technology to program flexible and efficient distributed applications. Most current systems implement semantics that are hard if not impossible to prove correct. In this paper we present Mob, a scripting language for Internet agents encoded on top of a process calculus and with provably sound semantics.

1 Introduction

This paper presents a scripting language, Mob, for programming mobile agents in distributed environments. The semantics of the language is based on the DiTyCO (Distributed TYped Concurrent Objects) process calculus [1], whose implementation provides the run-time for the language. In particular, we rely on it for interprocess communication and code migration over the network.

The development of mobile agents requires a software infrastructure that provides migration and communication facilities, among others. Current frameworks that support mobile agents are mostly implemented by defining a set of Java classes that must then be extended to implement a given agent behavior, such as Aglets [2], Mole [3], or Klava [4]. Mob, on the other hand, is a simple scripting language that allows the definition of mobile agents and their interaction, an approach similar to D'Agents [5]. However, Mob applications may interact with external services programmed in many languages such as Java, C, TCL or Perl. The philosophy is similar to that used in the MIME type recognition. The runtime engine matches a file type against a set of internally known types and either launches the correspondent application or simply executes the code.

Mob also differs from other mobile agent frameworks in the fact that it is compiled into a process-calculus core language, and its semantics can be formally proved correct relative to the base calculus. Thus, the semantics of Mob programs can be understood precisely in the context of concurrency theory.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 40–43, 2003.

2 Language Description

In this section we describe the basic language abstractions.

2.1 Agents

The Mob programming language is a simple, easy-to-use, scripting language. Its main abstractions are mobile agents that can be grouped in communicators allowing group communication and synchronization. Programming an agent involves implementing two steps: the **init** and the **do** methods. Method **init** consists of a setup that will execute before the agent starts its journey. It is usually used to assign initial values to the agent's attributes. The **do** method defines the agent's actions/behavior on the journey.

Agents have several built-in attributes: **owner** carrying the identification of the agent's owner; **itinerary** carrying the agent's travel plan; and **strategy** carrying the definition of a strategy of how the itinerary must be traveled. The programmer may define as many new attributes as he wishes. Their usefulness is to hold state to be retrieved when the agent migrates back home. The following example presents the skeleton of a Mob agent definition, Airline. Beside the built-in attributes, a new one, named price, has been defined.

```
agent Airline {
    price;
    init { price = 0 }
    do { // Implementation of the agent's actions/behavior }
}
```

Once the template for an agent is defined an undetermined number of agents can be created. The following example creates an agent named airline owned by johndoe and with **home** hosts host1 and host2. One can also launch several agents at once using the **-n** flag. airlineList will contain the list of the agent's identifiers. Notice that the attribute initialization supplied in the agent constructor will not override the ones in the **init** section.

```
airline = agentof Airline -u "johndoe" -h "host1 host2"
airlineList = agentof Airline -n 10 -u "johndoe" -h "host1 host2"
```

Each agent must be associated to an owner, defined in an entry of the Unix-like file named passwd. An entry for each user contains login, name, password and group membership information. Following the Unix policy for user management, users may belong to groups defined in the groups file, sharing their access permissions. Each Mob enabled host must own both files in order to authenticate each incoming agent. As featured in FTP servers, an agent can present itself as anonymous for limited access to local resources.

2.2 Communicators

Communicators are conceptually equivalent to MPI communicators [6] and allow group communication and synchronization. The **communicator** construct requires the list of agents (eventually empty) that will start the communicator. Other agents may join later.

2.3 Instructions

Most of the statements included in Mob are common in all scripting languages (**for**, **while**, **if**, **foreach** and **switch**), the only difference lies in the **try** instruction, a little different from the usual error catching instructions found in, for instance, TCL. Its syntax is similar to the try/catch exception handler instruction of Java, allowing specific handling of different types of run-time system exceptions. Mob provides instructions to define a mobile agent's behavior and its interaction with other agents and external services. These commands can be grouped in the following main sets:

1. agent state (**email**, **owner**, **home**, **itinerary**, **strategy**, **hostname**, etc) and creation (**clone**).
2. mobility: **go**.
3. checkpointing: **savestate** and **getstate**.
4. inter-agent communication: asynchronous (**send**, **recv**, **recvfrom**), synchronous (**bsend**, **brecv**, **brecvfrom**), communicator-wide (**csend**) and multicast (**msend**). There are variants of these functions for use with the HTTP and SMTP protocols (e.g., **httpcsend**; **smtprecv**). These variants are useful to bypass firewalls that only allow connections to ports of regular services.
5. managing communicators: **cjoin** and **cleave**.
6. execution of external commands: **exec**. This functionality allows the execution of commands external to the Mob language. The Mob system features a set of service providers that enable communication through known protocols, such as HTTP, SMTP, SQL and FTP. The interaction with these providers is possible through **exec**'s protocol flag.

Mob's input/output instructions are implemented as a syntactic sugar for the **exec** instruction. **open** filename could also be written as **exec** -p fs open filename. Mob features all the common file input/output commands.

3 Programming Example

The example defines a network administration agent that performs a virus check by perpetually moving through the hosts. If a virus is found and the installed version of the anti-virus cannot solve the problem, the agent updates the anti-virus software and retries to remove the virus. If the problem remains an email is sent to the administrator. This implementation considers that all the anti-virus software is present, thus not protecting the **exec** calls with **try** instructions.

```
agent Antivirus {
    init { strategy = "circular" }
    do {
        if ([exec viruscheck] == -1) {
            version = exec viruscheck -version
            host = hostname
            try {
                go -h updates
```

```
                update = exec getupdate version
                try {
                    go -h back
                    exec applyupdate update
                    if ([exec viruscheck] == -1)
                        exec -p smtp email "Cannot erase virus on " + host
                } catch exec -p smtp email "Cannot connect to repository"
            } catch {
                exec -p smtp email "Cannot connect to " + host
                go -h back
            }
        }
    }
}
agentof AntiVirus -i "host1 host2" -u "Admin" -e "admin@mob"
```

When the agent reaches the end of the **do** code it migrates to the next host resulting of the application of the strategy over the itinerary. In this case, a circular strategy that moves accross all nodes of the itinerary forever. Although Mob features several strategies, it allows the programming of new ones. An agent's itinerary is seen as an object that can be managed through an iterator. The methods fand **previous** of the iterator define the Mob traveling strategy.

4 Future Work

Mob is currently under implementation. Future work will focus on features such as the implementation of a local exception handling mechanism and the development of external services, such as, recognition/execution of programs in several high-level languages (e.g. Java, C, TCL).

Acknowledgments. This work is partially supported by FCT's project MIMO (contracts POSI/CHS/39789/2001).

References

1. V. Vasconcelos, L. Lopes, F. Silva: Distribution and Mobility with Lexical Electronic Notes in Theoretical Computer Science 16(3), Elsevier Science Publishers, 1998.
2. Lange D.: Java Aglet Application Programming Interface. IBM Tokyo Research Laboratory. 1997.
3. Straber K., Baumann J., Hohl F.: Mole – A Java Based Mobile Agent System. Inst. for Parallel and Distributed Computer Systems, University of Stuttgart. 1997.
4. Bettini L., De Nicola R., Pugliese R.: Klava: a Java Framework for Distributed and Mobile Applications. Software – Practice and Experience, 32(14):1365–1394, John Wiley & Sons, 2002.
5. Gray R.: Agent TCL: A flexible and secure mobile-agent system. Department of Computer Science. Dartmouth College. 1996.
6. MPI Forum: The MPI Message Passing Interface Standard. www-unix.mcs.anl.gov/mpi/. 1994