Using Tcl Mobile Agents for Monitoring Distributed Computations

Dilyana Staneva, Emil Atanasov

Abstract: *Agents, integrating code and data mobility, can be used as building blocks for structuring distributed computations. Despite its attractiveness, mobile agent programming is still a disputable technique and needs further study. This paper reflects our experience in using mobile agents for implementing monitoring tasks. It gives a brief overview of the mobile-agent model of computations and the D'Agents platform in particular. Then, a monitoring application, built on top of D'Agents under Linux, is described. Features of mobile agent programming with D'Agents are discussed.*

Keywords: *distributed computing, mobile agent-based computing, monitoring*

INTRODUCTION

Mobile agents have been quite popular in academia and industry over the last 10 years. Numerous mobile-agent platforms, both academic and commercial, have been designed and developed.

A mobile agent encapsulates the ability to transfer *data and code* from one network location to another. It can be used as a building block for structuring distributed computations. The very concept of code mobility is not novel. Agent migration can be viewed as an evolution of process migration in a heterogeneous setting. Mobile agents typically "reside" in the middleware, not the operating system level. The novelty of the mobile-agent approach lies in its *high-level and flexible style*, integrating code and data mobility, suitable for organizing distributed applications.

1 MOBILE AGENTS - PROS AND CONS

In the context of the mobile-agent framework, an application is structured as a collection of mobile agents. In most cases, a mobile-agent solution can have a static counterpart, based on message-passing. Fig.1 shows two scenarios for accessing a remote resource: in the traditional request/response style, and in the mobile-agent style, in which an agent *migrates* to the location of the resource.

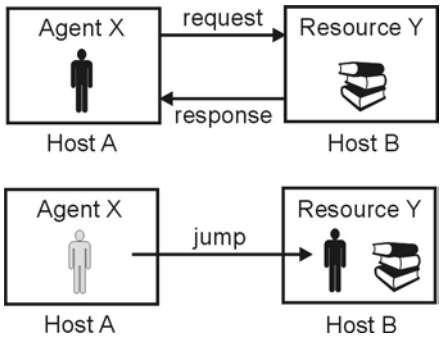

Fig.1 Stationary vs. mobile-agent solution

The motivation behind migration is the *locality of reference*. Mobility can be useful when an agent processes large volumes of data at the remote site, filters or monitors them locally and returns just the final results to its dispatching host, thus compensating for migration overheads.

The advantages of using mobile agents [4, 5], instead of traditional remote communication techniques, can be quantitative and qualitative. Migration, combined with locality of information processing, can improve application performance and scalability, and reduce network traffic. Therefore, one of the preferred application domains for mobile agents

is distributed information retrieval. A notable example is the StormCast weather-monitoring system, built on top of the Tacoma mobile-agent platform, which was codeveloped by the University of Tromsø and Cornell University [7].

An important qualitative aspect of mobile agents is their capability of disconnected operation. An agent is quite independent from its dispatching host: it can continue functioning even if its "home" is unavailable or unreachable, and communicate with it upon reconnection. Mobile agents are an attractive option for implementing "pluggable" services. A server, in the traditional client-server paradigm, exposes some fixed functionality, and a client-server based system is generally hard to reconfigure or extend. While a mobile agent, migrating to the server and "carrying" its code and data with it, can provide a flexible server-side extension. Mobile agents can significantly facilitate on-the-fly software deployment.

Despite its attractiveness, the mobile-agent paradigm has received a lot of criticism. Its advantages over traditional approaches are questioned and it is often viewed as yet another model of computations. There seem to be more mobile-agent platforms compared to real-world applications built with them, to reveal convincingly the strengths of mobile agents and outline the application areas best suited for them. There is still work to be done in building, experimenting with and evaluating mobile-agent solutions. The migrating nature of agents visiting different hosts imposes reasonable security concerns and resource consumption requirements.

Most mobile-agent platforms *add* mobility to an existing programming language, supporting a single or multiple instrumental languages. An *agent platform* therefore is basically composed of an *agent server* and an *execution environment* for each supported language (Fig.2). Platforms are generally oriented to interpreted languages. An agent "lives" as a process or a thread in the context of the agent server. An agent server must run on each machine willing to host mobile agents.

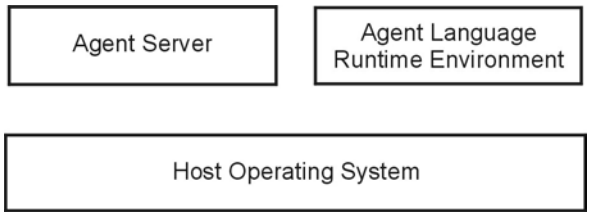

Fig.2 Basic structure of a mobile-agent platform

Mobile-agent platforms are broadly divided into Java-based (IBM Aglets [1], IKV++ Grasshopper [3]) and non-Java-based, generally using some scripting language, like Tcl or Perl (D'Agents [2, 4], Tacoma [7]). A mobile agent is programmed as a Java class or a Tcl script enriched with migration and communication capabilities. From the programmer's point of view, migration is generally expressed as a single statement, hiding lower-level details.

Java is by far the preferred language for mobile agents due to its built-in security, serialization/deserialization and remote method invocation support. We've used the Grasshopper platform to monitor the activity of Internet servers [6]. Our experience with it has shown that it is easy to comprehend and build extensible and reusable applications. Due to its high-level of abstraction and the expressiveness of Java, it can be an useful tool in teaching students the specifics of distributed computing.

When building distributed applications, we cannot always assume the presence of a Java virtual machine. Furthermore, Java-based mobile-agent platforms generally support "weak" mobility (an agent's *execution state* is not preserved during migration), and have no provisions for imposing limitations on resource comsumption (memory and CPU usage).

Here, we are focusing on Unix-based D'Agents, as a representative of non-Java-oriented mobile-agent platforms. The public v.2.0 release of D'Agents (former Agent Tcl) [2, 4] supports

Tcl/Tk. It provides "strong" migration, reduced to a single command, which transparently captures the current state of an agent, determined by the values of its variables and its execution stack, transfers it to a remote machine and resumes its execution there. The layered architecture of D'Agents is illustrated in Fig.3.

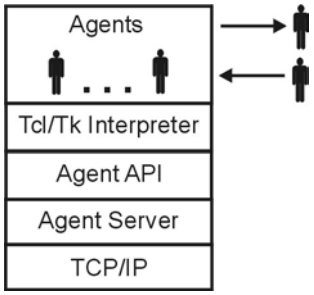

Fig.3 The architecture of D'Agents

The agent server provides basic functions for agents administration, migration and communication. An agent's identification is composed of the IP address of its current host, an unique integer, assigned to it by the agent server, and, optionally, an unique agent name. The core agent server functionality is extendable in the form of stationary service agents. Most notably, resource managers regulate access to local resources (e.g., the local file system).

Agent API is a language-independent core which supports the agent-server interaction. In the context of D'Agents, a mobile agent is a Tcl script, enriched with agent commands for migration, registration with the server and communication. Extended Tcl is executed by a modified Tcl/Tk interpreter.

2 THE MONITORING PROBLEM

A distributed simulation task is split within a cluster of cooperating Linux workstations. Each workstation runs a group of core simulation processes, as well as some logical processes, which execute their share of the particular simulation task. A key aspect of each simulation, especially in a distributed setting, is to monitor the "participants", both individually and as a whole.

Monitoring can be organized in a purely centralized or decentralized scenario, or a combination of these. We are focusing on open, flexible solutions, relevant to the distributed nature of monitored entities themselves - groups of processes on a network.

The monitoring problem can be solved traditionally, using some form of remote socket-based communication. But instead of retrieving information *remotely*, the components of the monitoring system could *migrate* to the very sources of information, processing and filtering data *locally*. Thus the monitoring problem fits "naturally" into the mobile-agent framework - the monitoring application, supporting the distributed simulation system, could be structured as a collection of mobile and stationary agents. Monitoring in our scenario is organized using two basic agent types - mobile *information agents* and stationary *control agents*, on top of the D'Agents platform v.2.0 under Linux. We explore the qualitative effects of using Tcl mobile agents for monitoring purposes.

3 THE MOBILE AGENT SOLUTION

We've developed a simple monitoring application with a single stationary control agent, dispatching one mobile information agent to visit all workstations involved in the distributed simulation task (Fig.4). A natural "parallel" enhancement of this application is to place one mobile agent per workstation.

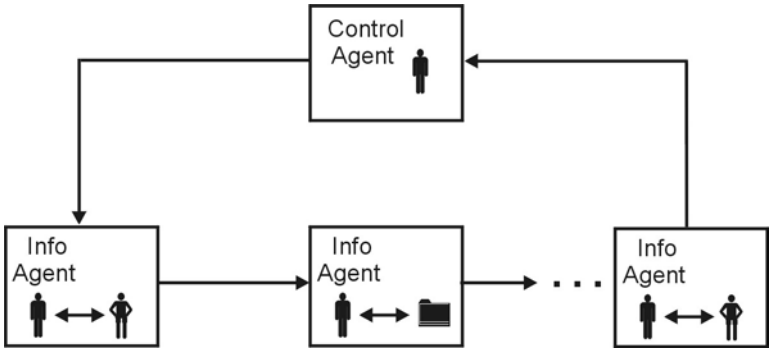

Fig.4 The mobile-agent monitoring scenario

Two kinds of *info agents* are provided: *specialized*, which monitor the simulation processes, and *general*, which retrieve information about the workstation average load (memory and CPU usage). The specialized info agent interacts locally with the simulation processes. When designing their interaction, we've aimed at imposing a minimum burden on the simulation processes, not "distracting" them from their main task. To achieve this, we've decided to organize local communication, using asynchronous UNIX sockets, instead of applying other IPC alternatives (e.g., message queues). Local interaction is performed in two steps. The info agent first tries to communicate directly with the simulation processes. In case the monitored party is unwilling or unable to communicate (e.g., it's not alive), the agent tries to retrieve information stored by the simulation processes in a local log file.

The *control agent* provides the user interface of the monitoring application and manages info agents. The user selects the type of info agent (specialized or general) and specifies its itinerary as a list of machines to visit. The control agent creates a "jumping" multi-hop agent, passing it the list of IP addresses. After which, it waits, using the blocking **agent_receive** command, until the info agent returns back the collected data as a single message:

```
agent_receive msgcode msgbody -blocking
```

The *specialized info agent*, communicating with simulation processes, is a simple Tcl procedure, a fragment of which is shown next:

```
foreach m $pc {
    if {[catch "agent_jump $m" result]} {
       append txt "Unable to jump to $m because of $result\n"
    }
    else {
       append txt "Jumped successfully to $m\n"
       agent_name monitor
       # do actual monitoring
    }
}
```

The info agent cycles through the list of machines **pc**, trying to jump to each of them. Agent migration is performed using the **agent_jump** command. On success, the agent jumps to the agent server on the specified machine (**m**). The agent server constructs a new agent, which continues its execution from the command following **agent_jump**. An agent "loses" its identification during migration, therefore it requests a new name from the agent server using **agent_name**. The result of **agent_jump** is analyzed via the Tcl **catch** command.

The monitoring agent logs its activity in a local file. It notifies simulation processes of its arrival and requests information from them. Communication, based on asynchronous UNIX sockets, is implemented with a helper C program. The monitoring application exploits the "strong" migration supported by D'Agents.

The D'Agents agent server offers basic functionality for managing migration and communication. It can be easily extended in the form of service agents. We've developed a stationary *"status" agent*, which reports information about agents residing on the local server.

The complementary C program has to be delivered "on the fly" to all workstations to be visited. Instead of installing it manually, we've developed a mobile *"installer" agent*, using an alternative D'Agents communication pattern, suitable for transferring files. The "installer" jumps to the remote machine and tries to set a communication channel (a "meeting") with its dispatching agent **agent(root)**:

```
set sock [agent_meet $agent(root)]
```

When the socket connection between the two agents is established, the installer receives the executable file:

```
meeting_getfile $sock $fd -blocking
```

and ends the meeting with

```
meeting_close $sock.
```

The D'Agents platform provides several security mechanisms. The local agent server and the individual agents run as processes owned by a non-privileged Linux user. Migration range is restricted with a configuration file, containing all IP addresses, from which the local agent server is willing to accept agents. Local resources consumption is regulated by resource managers.

CONCLUSIONS AND FUTURE WORK

The monitoring application we've described, is a sample, not a real-world application, which could clearly identify the strengths and weaknesses of mobile agents. Nevertheless, our experience with the general mobile-agent framework and mobile-agent programming on top of D'Agents provides knowledge of their usefulness and convenience.

Mobile-agent platforms in general are quite *easy to learn and use*, compared to lower-level remote communication mechanisms. The D'Agents platform supports various primitives for migration and communication via individual messages or a stream of messages, at different levels of abstraction and complexity. This diversity greatly facilitates the application developer. Furthermore, mobile-agent platforms can be used for rapid prototyping, to enable choosing the best suited structuring technique (mobile or non-mobile) for distributed solutions. Due to its high level of abstraction and flexibility, mobile agent-based programming can be a valuable tool in teaching distributed computing.

Existing platforms offer basic functionality, which is readily extendable in the form of *service agents*.

The Tcl language is quite restricted with respect to data type and code modularization. The limitations of Tcl can be compensated by using host operating system commands and custom programs. Thus, Tcl can be used for managing mobile agents, while their actual functioning could be programmed in a more expressive language, like C.

The mobile agent framework accommodates a broad domain of applications. In our simple monitoring scenario, we've implemented both monitoring and dynamic components deployment *uniformly*, within the same framework, instead of a combination of traditional approaches. We consider a very attractive feature of mobile agents their "pluggability". A

mobile agent can be replaced or extended, thus providing new functionality on migration.

Mobile agent platforms support basic security mechanisms, yet security is a primary concern in mobile agent-based computing. Our future efforts will be oriented towards securing agents and agent hosts.

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. C. Harrison, D. Chess and A. Kershenbaum. Mobile agents: Are they a good idea? Research Report, IBM Research Division, 1995.

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

7. Tacoma Mobile-Agent System. http://www.tacoma.cs.uit.no

ABOUT THE AUTHORS

Dilyana Violinova Staneva, Assistant Professor, PhD. Computer Science & Engineering Department, Technical University of Varna, Bulgaria. Phone: +359 52 383 ext. 424, Å-mail: dilyana@linux.tu-varna.acad.bg.

Emil Krasimirov Atanasov, Master student. Computer Science & Engineering Department, Technical University of Varna, Bulgaria.