Monitoring Mobile Code[1]

Fernando Luís Dotti, Lucio Mauro Duarte
fldotti@kriti.inf.pucrs.br, lmduarte@zaz.com.br

Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681 - Prédio 16 - CEP 90619-900 - Porto Alegre – RS – Brasil
Fone/fax: +55 51 320-3611

Abstract

Code Mobility brings the possibility of building distributed systems better suited to a range of application areas. While platforms supporting Code Mobility are already in use, support is still lacking for developing and managing mobile code systems. Mobile code monitoring is proposed in this paper as means to better identify application areas suited to code mobility, to test mobile applications, and to achieve mobile code accounting and performance management. Mobile code monitoring is then discussed, the main problems involved are identified and approaches for their solution too. An example of a mobile code monitor implementation is given, its functionality and architecture are explained and plans for further developments are outlined.

Key-words**:** *Distributed Systems, Mobile Code, Monitoring, Accounting, Software Development*

1 Introduction

In the last years, code mobility emerged as a new approach for building distributed applications. This new approach is based on the idea that a software component, physically located in a network node, is able to, dynamically, move through the network, migrating from node to node. After [1], with the code mobility characteristics, new design paradigms for distributed software emerged as option to the classical client/server one: remote execution, code on demand, and mobile agents.

A wide field of applications is foreseen to be scenario for mobile code applications. Network management [1], electronic commerce [5], distributed information retrieval [3], dvanced telecommunication services [4] and active networks [12], [13] are among them.

Although platforms supporting code mobility are already in use (e.g. Voyager [6] and Aglets [8]), code mobility is still recent and appropriate support is lacking. Developes have few or no tools to debug/test if a mobile component is exhibiting a mobile behavior as expected or if a mobile application really has advantages if compared to an analogous fixed one (e.g. remote communication, processing, security, etc).

One approach to address such aspects is to enhance the functionality offered by mobile code support platforms. A key action to test, account and verify performance of mobile code applications is to monitor such applications. This paper proposes to enhance existing mobile code environments to support monitoring functions.

In the next section, the text discusses the main aspects involved in monitoring mobile code, what could be monitored and how to monitor it. In section 3 the architecture and implementation of a mobile code monitor is discussed, its functionality and performance data are presented. In section 4 conclusions and future work are outlined.

2 Monitoring Mobile Code

The discussion around monitoring mobile code can be organized in two parts: what to monitor and how to do it. Both are discussed in the next sections.

[1] Project partially supported by FAPERGS.

2.1 What to monitor ?

One could list items of interest to be monitored in a mobile code system. The following is a list of items per mobile code component:

- **Life cycle:** place and time of creation and finalization;
- **Migration:** departure and arrival times of a component exiting and entering different places;
- **Service usage:** amount of accesses to services, types of services used;
- **Hierarchy:** association of a component with its creator component.

By monitoring those items above, it is possible to derive information for the software developer or system manager. Some of them are:

- Total transfer time of a component;
- Total lifetime of a component;
- Total execution time of a component;
- Trace of the path described by a component;
- The creation relationship tree of an application (hierarchy of an application). This tree describes the sequence of creation of mobile objects. It is generated identifying mobile objects' parents, that is, which mobile objects created the others. The root of the tree is the first object created by the application, its children objects (objects created by it) are in the second level, their children objects in the third level and so on.

2.2 How to monitor ?

In order to establish a monitoring system to provide the information above listed with reliable results, several functions should be supported by mobile code platforms:

- **Unique mobile component identification:** the monitor should probide a unique identification to mobile objects in a distributed environment in order to be possible identify all of them even after many migration processes. This is necessary because some mobility platform, when a migration process starts, finalize the moving object in the origin and recreates it in the destination, changing its original internal identification;
- **Synchronization among the nodes:** nodes have to be synchronized to be possible to obtain a true time relation between various nodes in a distributed environment. This is essencial to correctly presenting the life cycle of mobile codes;
- **Life cycle and mobility event report:** most platforms for mobile code (e.g. Aglets, Voyager) generate events related to the mobile behaviour of the components. The monitor should gather those reports;
- A **service usage accounting function** should be also provided, a register containing all interfaces used by a component and reported events as well.

Besides that support, monitoring mobile code in a distributed infrastructure - scalability

3 Implementing a Mobile Code Monitor

In this section the architecture and implementation of a mobile code monitor is presented. The monitor was called Mobile Object Monitor (MOOM).

3.1 Scope

The MOOM is a tool for monitoring mobile codes and was developed to give to its users important information about their mobile applications behaviour. It takes profit of events and runtime information supported by the mobility platform to create reports containing such information about monitored events

(see 2.1). The information obtained by the monitor from the platform is related to the life-cycle of the components and their mobility:

- Time and place of creation of a mobile component (or mobile object);
- Time and place of finalization (garbage collection) of a mobile component;
- Time of departure from a place and the destination place the component migrates to;
- Time of arrival in a place.

With those information and some functions provided by itself, MOOM is capable to present aggregate information, as those listed in item 2.1.

3.2 Architecture of MOOM

In each monitored place, a representative module of MOOM, named Secondary MOOM, runs to gather the events generated by the mobility platform. The monitoring station of the monitoring environment is configured with a module named Primary MOOM, which supports queries to obtain the information listed in the last item. The Primary MOOM and the Secondaries MOOM connected to it creates the definition of *monitoring domain*. That is, a domain is a group of Secondaries MOOM connected to a Primary MOOM. An example of domain and basic architectures of Primary and Secondary MOOM can be seen in figure 1.

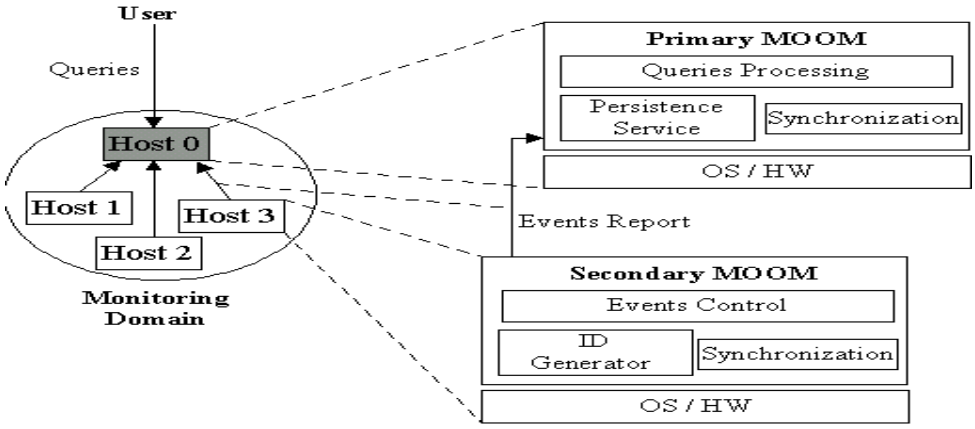

Figure 1. Ilustration of a domain with 3 Secondaries MOOM.

During execution, the Secondary MOOM awaits for events to be reported by the platform. Those events are received by a pool of threads. When one such thread receives an event, it adds a timestamp and dispatches it to the Primary MOOM. When a component (object) is created, a unique identifier is generated for that component and it is marked with the identification of the parent component. The Primary MOOM receives the events, organizes them in a database, and supports queries on that data.

According to those aspects discussed in 2.2, the creation of a unique identifier is provided by a MOOM function. Another MOOM function, based on Cristian's algorithm [8], is used to synchronize all nodes involved in the monitoring environment (domain). Life cycle and mobility events are, as said before, obtained from the mobility platform. However, MOOM is not platform dependent because its architecture was developed to easily be adapted to any mobility platform. The modification necessary to adapt it for others mobility platforms is restricted to a component that is responsable for obtaining events reports from the platform. Those events are obtained according to the platform and some modifications in this component can make possible monitoring events from well-known platforms. No other component in the MOOM architecture is modified or affected by a platform change.

3.3 Practical results

MOOM [10] is implemented in Java™ [9], [11], and considers the use of ObjectSpace Voyager™ as mobility support platform, although it can be used with any other platforms offering basic monitoring capabilities. The Primary MOOM uses JDBC to connect to a database to store reported events and retrieve event information when required.

One could download MOOM classes, examples of applications and documentation and know more about it in the site http://www.inf.pucrs.br/~fldotti/monitor.

Some tests were made to figure out if MOOM is really reliable and how much its affects the monitored applications performance. Results are presented in table 1. The application executed during tests is called Ping Pong and it creates a mobile object that moves itself between two nodes. This object goes from one to the other node and turns back. Each go-and-come movement counts a lap. Times are presented in seconds.

	Without MOOM		**With MOOM**	
	1st Exec.	2nd Exec.	1st Exec.	2nd Exec.
Execution total time	40	29.2	57.24	43.6
Object creation time	1.37	0.55	1.76	0.7

Table 1. Comparative times of Ping Pong application with and without MOOM (100 laps).

4 Conclusions and Future Work

In this paper, we have presented an architecture for mobile code monitoring. The practical results show that the monitor can be well used in a test environment to retrieve the mobile behavior of applications. This should help developers in debugging mobile applications.

A future work direction is to enhance the functionality offered by the prototype. Services and computational resources usage should be also monitored, and the cooperative interface among monitored domains implemented.

5 References

[1] FUGGETA, Alfonso, PICCO, Gian Pietro, VIGNA, Giovanni. Understanding Code Mobility. **Transactions On Software Engineering.** IEEE , v.24, 1998.

[2] CARZANINGA, Antonio, PICCO, Gian Pietro, VIGNA, Giovanni. **Designing Distributed Applications With Mobile Paradigms.** [s.l.], [199-].

[3] KNUDSEN, P.. **Comparing Two Distributed Computing Paradigms – A Performance Case Study.** M. S. thesis, University of Tromso, 1995.

[4] MAGEDANZ, T., ROTHERMEL, K., KRAUSE, S.. **Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?.** IEEE, Berlim, 1996.

[5] MERZ, M., LAMERSDORF, W.. Agents, Services and Eletronic Markets: How Do They Integrate?. In **Proc. of the Int'l Conf. On Distributed Platforms.** IFIP/IEEE, 1996.

[6] **http://www.objectspace.com/products/prodVoyager.asp**

[7] **http://www.trl.ibm.co.jp/aglets/index.html**

[8] TANENBAUM, Andrew S.. **Modern Operating Systems.** Prentice-Hall International Editions, EUA, 1992, pp. 463-475.

[9] CORNELL, Gary, HORSTMANN, Cay S.. **Core Java.** Makron Books, São Paulo, 1997, 807 p.

[10] DOTTI, Fernando Luís; NYGAARD, Fernando; DUARTE, Lucio Mauro; REZNICEK, Roberto Domingues. MOOM - Um Monitor de Objetos Móveis . In **25th Latinoamerican Conference on Informatics.** Asuncion, Paraguay, august 30th to september 3th, 1999. (Portugese)

[11] LINDEN, Peter van der. **Just Java.** Makron Books, São Paulo, 1997. 543 p.

[12] Tennenthouse, David L.; Wetherall, David J.; Smith, Jonathan M.;Minden, Gary J.; Sincoskie, W.D.. **A Survey of Active Network Research.** IEEE Communication Magazine, January 1997.
[13] Smith, Jonathan M.; Calvert, Kenneth L.; Murphy, Sandra L.; Orman, Hilarie K.; Peterson, Larry L. **Activating Networks: A Progress Report.** IEEE Computer, April 1999.