Mobile Code Paradigms and Technologies: A Case Study

Carlo Ghezzi and Giovanni Vigna

Dip. Elettronica e Informazione, Politecnico di Milano
P.za L. Da Vinci 23, 20100 Milano, Italy
`[ghezzi | vigna]@elet.polimi.it`.

Abstract. The opportunities offered by the Internet are encouraging research aimed at the creation of a computational infrastructure that exploits the wide spread communication infrastructure. The mobile computation paradigm is a proposal to build a computational infrastructure that goes beyond the well-known client-server paradigm and increases dynamicity and flexibility. Despite the promising first steps, there is still confusion on the role of paradigms and technology in the development on applications based on the mobile computation paradigm. We present a case study in which we develop several versions of an application using different paradigms and different technologies in order to show when these concepts come into play and which are their relationships.
Keywords: mobile code, design paradigms, case study

1 Introduction

The Internet has now become the largest distributed system ever built. While the deployed *communication infrastructure* is evolving at a fast pace providing larger bandwidth by means of new network technologies and protocols, the *computational infrastructure* is still rather primitive and only recently has become the focus of systematic research activities. By computational infrastructure we mean both the *technologies* available to implement and support execution of network-centric applications and the design *paradigms* according to which such applications can be structured. Although application developments are literally exploding on the Internet, such applications are mostly developed using unstable and evolving technologies, and very little is known regarding design paradigms, besides the conventional client-server model.

To consolidate the explosion of network-centric applications, we believe that research is needed both at the level of enabling technologies and at the level of design paradigms. In fact, in order to fill this gap, the concept of *mobile computations* has become the focus of much recent research [1]. In this framework, the computational infrastructure is composed of a (possibly world-wide) distributed environment with several *computational environments* (CEs) that support the computations of *executing units* (EUs). Such EUs may change their computational environment and even the code they execute dynamically. Several technologies, like Telescript [16], Messengers [15], and Java [10] have been proposed. Still, the development of applications based on the mobile computation paradigm (shortly, *Mobile Code Applications* or MCAs) lacks a methodological background.

We envision an idealized process by which software designers are equipped with methods that allow them to select the most appropriate design paradigm for any specific application. The software architecture designed according to the selected design paradigm should then be mapped onto an implementation using the technology (languages and their support tools) that best fit the chosen paradigm.

Unfortunately, the distinction between paradigms [4] and technologies [5] is often blurred and not well understood. The focus of our work is exactly on understanding the conceptual foundations of mobile computations, trying to identify the specific issues that characterize mobile code applications at all stages of their development process. In particular, in this paper we address the issue of design paradigms and technologies. We identify a repertoire of paradigms that are suitable for several kinds of mobile code applications and we discuss their relationships with technologies. We will show how the different paradigms can be implemented using different technologies. We will also show that certain paradigms are supported more easily and efficiently by certain technologies. As a result, one might develop guidelines that help to choose the most appropriate technologies for each paradigm. This may be viewed as an initial step in the direction of the above mentioned idealized process.

The paper is organized as follows. In Section 2 we introduce a case study that will be used throughout the paper to discuss both possible design paradigms and implementation issues. The case study is the search of information in a distributed database. In Section 3 we identify an initial repertoire of different design paradigms for mobile applications. Based on such paradigms, Section 4 presents different design alternatives for our case study. In Section 5 we present some classes of technologies, while in Sections 6, 7, 8, we shortly present the different implementations of our application. In Section 9 we discuss implementation issues. Section 10 draws some conclusions and illustrates future work.

2 A Case Study

Our study application is a distributed information retrieval system. The application is composed of several *Database Management Systems* (DBMSs) and of a *Search Engine*. DBMSs are distributed over a set of computational environments. They are identified by a location, i.e., the computational environment they are running on, and a symbolic name that uniquely identifies a particular DBMS in the corresponding CE. Each DBMS manages a single table composed of two fields: *keyword* and *data*. The keyword is an identifier of an entity. The data field may either be some kind of information related to the entity denoted by the keyword or the identifier of another database that may contain additional information about the corresponding keyword.

The Search Engine's task is to gather the largest amount of information associated with a particular keyword, given an initial set of DMBSs. If during the query of these initial DBMSs some references to other DBMSs are found, the Search Engine continues the search on those DBMSs.

3 Mobile Code Paradigms

The design of MCAs is a challenging task. Such applications are highly dynamic from the point of view of both code and location and therefore it is necessary to take into account these concepts at the design level. We identified three main paradigms for mobile computations: *Remote Evaluation* (REV), *Code on Demand* (COD), and *Mobile Agent* (MA) [4]. Although we don't claim that such paradigms cover all possible design structuring styles for network-centric applications, they can be viewed as the most typical representatives.

Given two interacting components A and B of a distributed architecture, these paradigms differ in how the *know-how* of the application, i.e., the code that is necessary to accomplish the computation, the *resources*, i.e., the inputs/outputs of the computation, and the *processor*, i.e., the abstract machine that executes the code and holds the state of the computation, are distributed between the components. In order to let this computation to take place, all the three capabilities above have to be present at one location *at the same time*. We assume A to be the entity that causes the interaction and the one that is interested in its effects. Table 1 shows how the paradigms behave with respect to the capabilities described above. The table also lists the *Client-Server* (CS) paradigm. Although conceptually CS cannot be viewed as a paradigm for mobile computations (no mobility takes place), we included it because it is a highly used paradigm on network-centric applications.

Paradigm	*A side*	*B side*
client-server	–	know-how resources processor *
remote evaluation	know-how	resources processor *
code on demand	* resources processor	know-how
mobile agent	know-how processor	resources *

Table 1. Mobile code paradigms. This table shows the location of the capabilities just before the interaction takes place. The star (*) indicates where the capabilities meet, i.e., where computation associated with the interaction takes place.

In the CS paradigm, a server component (B in Table 1) exports a set of services. The code that implements such services is owned by the server component, thus, we say that the server holds the *know-how*. It is the server itself that executes the service, thus it has the *processor* capability. The client (A in Table 1) is interested in accessing some entity managed by the server, and therefore it is the server that has the *resources*.

In the REV paradigm, the executor component (B) offers its computational power (the *processor*) and its *resources*, but does not provide any "specific" service. It is A that sends the service code (the *know-how*) that will be executed by B in its location.

In the COD paradigm, component A initially is unable to execute its task. It is B that provides the code (i.e., the *know-how*). Once the code is received by A, the computation is carried out on A's location, thus, A holds the *processor* capability. The computation involves only local files and local devices; thus, A holds the *resources*.

In the MA paradigm, A has the *know-how* and *processor* capabilities. The computation associated with the interaction takes place on B's location where the *resources* involved reside.

4 Designing an Application

Usually, an application (or parts thereof) may be designed following different paradigms. In our case study, we have designed our application using the well-known Client-Server paradigm and two mobile code paradigms, namely, Remote Evaluation and Mobile Agent.

4.1 Client-Server Design

According to the CS paradigm, DBMSs play the role of servers while the Search Engine is a single component that acts as a client. Each DBMS offers a *query* operation that, given a particular keyword and an index n, returns the information contained in the n^{th} tuple containing the specified keyword or an error if such tuple does not exist. The Search Engine queries the DBMSs contained in its initial set and then updates such set with possible new databases whose identifiers have been found by previous queries. This process goes on until there are no more unqueried DBMSs.

4.2 Remote Evaluation Design

According to the REV paradigm, the Search Engine component sends a code fragment to the location of each DBMSs it has to query. Such fragments are evaluated on the remote locations where DBMSs are located, i.e., new components are created to execute the code on the destination sites. Such components query their local DBMSs using the same primitives that DBMSs export in the CS case. When a remote component terminates its task, it sends back to the

Search Engine the results of queries. If the results contain some references to other DBMSs new REV interactions with these locations are started.

4.3 Mobile Agent Design

Following the MA paradigm, the Search Engine is modeled as a series of components that roam through the locations of the DBMSs in order to collect information about the given keyword. Initially, as in the case of REV, a component is sent to the location of each DBMS of the given DBMSs set. Each component has an initially empty "to-be-visited" list containing the DBMSs that have to be searched and an empty "visited" list containing the DBMSs that have already been visited. On each location that hosts a DBMS there is a "history" list that each visiting component updates with the list of its visited DBMSs.

A component A migrates to the location of the first DBMS in its "to-be-visited" list. There, it performs the following actions:

1. it checks the local "history" and prunes its "to-be-visited" list accordingly;
2. it updates the local "history" with its "visited" list;
3. it checks if another component B is presently searching the local DBMS. If it is so, A interacts with B following these steps:
   (a) A provides B with its own "visited" list, so that B can update its "to-be-visited" list accordingly;
   (b) A balances its "to-be-visited" list with the B's one;
   (c) A removes the current DBMS from its "to-be-visited" list;
4. if its "to-be-visited" list does not contain the current DBMS anymore, A skips to step 7;
5. if the list still contains the current DBMS, it searches the DBMS, possibly enriching its "to-be-visited" list;
6. adds the visited DBMS to the local "history";
7. migrates to the location of the first DBMS of the "to-be-visited" list if any; otherwise it goes back to the starting location and reports its results.

5 Implementation Technologies

Having designed a MCA according to some paradigm, one has to choose a technology to implement it. Given a particular paradigm, which technology should be used?

We identify three classes of technologies [7]:

Message-based These technologies enable the communication between remote EUs in the form of message exchange. A typical example is RPC [2].

Weakly mobile These technologies provide mechanisms that enable an EU to send code to be executed in a remote CE together with some initialization data or to bind dynamically code downloaded from a remote EU. Examples of such technologies are the *rsh* facility in UNIX, languages like M0 [14] and Obliq [3], or systems like TACOMA [9] and Mole [13].

Strongly mobile These technologies enable EUs to move with their code and execution state to a different CE. An example is represented by the Telescript technology.

We implemented the different architectures of our application using different types of technologies. We chose Tcl-DP [12], Obliq [3], and Agent Tcl [8].

Tcl-DP is an extension of the Tcl language [11] for distributed programming. Tcl-DP provides support for TCP/IP and RPC programming. It is a message-based technology.

Obliq is an untyped, object-based, lexically scoped, interpreted language. It supports remote method invocation and remote evaluation of code. Therefore it is both a message-based and a weakly mobile technology.

Agent Tcl provides a Tcl interpreter extended with support for EU migration. An executing Tcl script can move from one host to another with a single `jump` instruction. A `jump` freezes the program execution context and transmits it to a different host which resumes the script execution from the instruction that follows the `jump`. Therefore Agent Tcl can be viewed as a strongly mobile technology.

Sections 6 through 8 provide an informal and succinct description of how the designs based on the CS, REV, and MA paradigms can be implemented using Tcl-DP, Obliq, and Agent Tcl.

6 Implementing the Client-Server Architecture

6.1 Message-based Technology

In this version of the application, we used Tcl-DP to implement the CS architecture. The DBMS component is implemented as a process running a Tcl-DP interpreter acting as an RPC server that exports a single *query* service with the characteristics described in Section 2. The Search Engine component is implemented as another process running a Tcl-DP interpreter that acts as an RPC client. The client queries the server using RPC primitives. The CEs are represented by UNIX hosts.

6.2 Weakly Mobile Technology

We used the Obliq language to implement this version of the application architecture. Even if Obliq enables the remote invocation of methods, we decided to use the mechanisms for remote evaluation of code offered by the language, since our goal here was to evaluate how a technology based on the remote evaluation of code supports the CS paradigm. We have implemented locations as Obliq interpreters that export *engines* (i.e., execution services), and DBMSs as objects that enclose the database data and provide a query method. The Search Engine is a thread in an interpreter that uses the *engines* exported by other interpreters in order to evaluate segments of code remotely. At run-time the Search Engine chooses a DBMS and sends a piece of code containing a single query method invocation to the remote engine of the interpreter containing the DBMS. The single invocation is performed in the remote interpreter by a newly created thread and then the results are delivered back.

6.3 Strongly Mobile Technology

We used Agent Tcl to implement the Client-Server architecture. In this setting the DBMSs are stationary agents that accept queries from agents located on the same site (represented by a UNIX host extended with the Agent Tcl run-time support). The Search Engine is a moving agent that, in order to query a DBMS, jumps to the corresponding site, performs a single query, and then jumps back to its starting site.

7 Implementing the Remote Evaluation Architecture

7.1 Message-Based Technology

We used the Tcl-DP language to implement the REV architecture. In this case, the remote site is extended with a process running the Tcl-DP interpreter that acts as a code executor. It is an RPC server that exports the service *execute*. Such service takes as a parameter a Tcl-DP script . DBMSs are Tcl-DP interpreters that export the *query* service as in the CS case, but only to local processes. When the Search Engine wants to evaluate remotely the code that performs a set of queries, it sends to the remote executor a service request containing the query script. In turn, the executor interprets the script that queries the DBMS and then returns the results back.

7.2 Weakly Mobile Technology

We used Obliq to implement the REV architecture. In this implementation, the Search Engine thread requests the execution of the query code to the remote engine corresponding to a particular DBMS. A new thread is created to execute the code. The newly created thread performs all the needed queries to the local DBMS, which, as in the case of CS, is an object owned by the remote interpreter. Then, the results are delivered back to the source site.

7.3 Strongly Mobile Technology

We used Agent Tcl to implement the REV architecture. In this implementation the Search Engine creates a set of agents that jump to the sites of each DBMS in the initial set, perform the set of query on the local DBMSs, and then return to the original site with the results. Such results are analyzed and then a new "squad" of agents is sent to the sites of the DBMSs whose identifiers have been found during the execution of previous query process.

8 Implementing the Mobile Agent Architecture

8.1 Message-Based Technology

We used Tcl-DP to implement the MA architecture. In this case, the DBMSs are RPC servers that accept queries from local processes while Search Engine mobile components are Tcl-DP scripts. In order to move from site to site, such mobile components pack their code and state (i.e., their "to-be-visited" and "visited" lists and the values of some state variables that keep trace of their computation) into a message and then send the message to an executor that unpacks the message and creates a new component. The new component uses the information stored in the state part of the message in order to restore its execution flow. After dispatching the message to the remote executor, the sender terminates.

8.2 Weakly Mobile Technology

We used Obliq to implement the MA architecture. Since Obliq threads cannot migrate from one interpreter to another, when a Search Engine component must migrate to another interpreter it creates a copy of itself on the remote site and then terminates. In order to keep track of the execution, an object is used to maintain the state of the component. After each "jump", the newly created thread must create a local copy of the object representing the state and perform some operations based on its contents in order to re-establish the state of the computation.

8.3 Strongly Mobile Technology

We used Agent Tcl to implement the MA architecture. The Search Engine is composed of a set of agents. At startup, such agents migrate to the locations of the given set of DBMSs. There, they perform the steps described in Section 4.3, i.e., they query the local DBMS and, if references to other DBMSs are found, they migrate to such locations, until their "to-be-visited" list becomes empty. Eventually, they jump back to their starting site and report their findings.

9 Discussion

The case study described in the previous sections shows that paradigms and technologies are not completely orthogonal. In principle, it is possible to implement applications developed with any paradigm by using any kind of technology, given that such technologies allow for the communication between EUs. However, we have found that some technologies are more suitable to implement applications designed using particular paradigms. Unsuitable technologies force the developer to program, at the application level, some mobility mechanisms or force an inefficient, counter-intuitive use of the existing ones.

As shown in Table 2, message-based technologies are well suited for implementing architectures based on the CS paradigm. If they are used to implement REV-based architectures, they force the implementor to use (unnaturally) code as data and to program the evaluation of such code explicitly. Even worse, if message-based technologies are used to implement MA-based architectures, the programmer has also to code state management, i.e., auxiliary variables must be used to keep the state of the computation and unnatural code structures must be used to restore the state of a component after migrating to a different site.

Weakly mobile technologies that allow to execute segments of code remotely are naturally oriented towards the implementation of MCAs designed according to the REV paradigm. These technologies are inefficient to implement CS architectures since they force the remote execution of segments of code composed of a single instruction. Therefore a new EU is created in order to execute this "degenerate" code. On the contrary, in order to implement MCAs based on the MA paradigm, the programmer has to manage, at the program level, the packing/unpacking of the variables representing the state and the restoring of the EU execution flow.

Strongly mobile technologies are oriented towards MA-based applications while they are not suited for implementing applications based on the CS and REV paradigms. In the former case, the programmer has to "overcode" an agent in order to have it moved to the server site, execute a single operation and jump back with the results. Such implementations could be rather inefficient since the whole EU state is transmitted back and forth across the network. In the latter case, in addition to the code to be executed remotely, the implementor has to add the migration procedures. Furthermore, the state of the EU is to be transmitted over the network.

Summing up, technologies may reveal to be too powerful or too limited to implement a particular architecture. In the first case resources are wasted, resulting in inefficiency. In the second case the programmer has to code all mechanisms and policies that the technology does not provide.

10 Conclusions

The ever-increasing growth of MCAs has stimulated research on methodologies and technologies that constitute the computational infrastructure of this new class of applications. Several technologies, with different characteristics have been proposed. Yet, a clear methodological framework is still missing; many concepts are not clearly defined and there is often confusion about paradigms and technologies

Mobile code paradigms are the styles according to which the software architecture of an application can be designed in terms of interacting components. Technologies provide mechanisms that can be used to implement network-centric distributed applications.

We have developed a case study in order to understand in which phase of the development these concepts come in play and which are the relationships

Technologies	Paradigms CS	REV	MA
Message-based	Well suited	Code as data Program interpretation	Code and state as data Program state restoring Program interpretation
Weakly mobile	Code is a single instruction Creates unnecessary execution units	Well suited	State as data Program state restoring
Strongly mobile	Code is a single instruction Creates unnecessary execution units Move state back and forth	Manage migration Move state back and forth	Well suited

Table 2. Relationships among paradigms and technologies.

between paradigms and technologies.

We have identified which class of technologies are well suited to implement applications designed using a particular paradigm and which are the difficulties or the inefficiencies that stem out from the use the inappropriate technology.

Our case study is a "toy" application of limited complexity. Future work could consider the study of a real-life mobile code application. In addition, more technologies and paradigms might be considered, other than those discussed here. At present, we are extending our work by developing a case study that includes the COD paradigm, which was not covered here, and uses other technologies like Java and TACOMA.

We did not cover phases of the development process other than design and implementation. As an example, testing and debugging phases are strongly influenced by the introduction of mobility. Testing of MCAs is much more difficult than testing of "traditional" applications since the configuration of components and the binding among components and code is dynamic. In addition, debugging code that is executed remotely is a challenging task. Further work is needed to fully understand the effect of the new computational infrastructure provided by the network on the entire lifecycle of MCAs.

Acknowledgments

A preliminary treatment of the issues discussed in this paper can be found in [6]. We wish to thank Tullio Dovera and Roberto Nespoli for implementing most of the experiments illustrated here. The previous work about classification of mobile code languages and paradigms was carried out with G. Cugola, A. Carzaniga, G. P. Picco, and A. Fuggetta.

References

1. J. Baumann, C. Tschudin, and J. Vitek, editors. *Proceedings of the 2^{nd} ECOOP Workshop on Mobile Objects ("Agents on the Move")*, 1996.
2. A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. Technical Report CSL-83-7, XEROX, October 1983.
3. L. Cardelli. Obliq: A language with distributed scope. Technical report, Digital Equipment Corporation, Systems Research Center, May 1995.
4. A. Carzaniga, G. P. Picco, and G. Vigna. Designing Distributed Applications using Mobile Code Paradigms. In *Proceedings of the 1997 International Conference on Software Engineering*, May 1997.
5. G. Cugola, C. Ghezzi, G.P. Picco, and G. Vigna. Analyzing Mobile Code Languages. In *Special Issue on Mobile Object Systems*, LNCS. Springer-Verlag, 1997. To appear.
6. T. Dovera and R. Nespoli. Paradigmi e tecnologie per lo sviluppo di applicazioni basate su codice mobile. Master's thesis, Politecnico di Milano, 1996.
7. C. Ghezzi, G. Cugola, G. P. Picco, and G. Vigna. A Characterization of Mobility and State Distribution in Mobile Code Languages. In *Proceedings of the 2^{nd} ECOOP Workshop on Mobile Object Systems*, July 1996.
8. R.S. Gray. Agent Tcl: A Transportable Agent System. In *Proceedings of the CIKM'95 Workshop on Intelligent Information Agents*, 1995.
9. D. Johansen, R. van Renesse, and F.B. Schneider. An Introduction to the TACOMA Distributed System - Version 1.0. Technical Report 95-23, "University of Tromsø and Cornell University", June 1995.
10. Sun Microsystems. The Java Language: A White Paper. Technical report, Sun Microsystems, 1994.
11. J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.
12. B. Smith and L. Rowe. Tcl-DP. Documentation, 1996.
13. M. Straßer, J. Baumann, and F. Hohl. MOLE: A Java Based Mobile Agent System. In *Proceedings of the $2^n d$ ECOOP Workshop on Mobile Objects ("Agents on the Move")*, 1996.
14. C. F. Tschudin. *An Introduction to the* MO *Messenger Language.* University of Geneva, Switzerland, 1994.
15. Christian F. Tschudin. OO-Agents and Messengers. In *ECOOP'95 Workshop on Objects and Agents*, August 1995.
16. James E. White. Telescript Technology: The Foundation for the Electronic Marketplace. Technical report, General Magic, Inc., 1994. White Paper.