

# Code Mobility Overview

(Essay for DIF 8914)

Li Jingyue

Jingyue@idi.ntnu.no

Department of Computer and Information Science (IDI),  
Norwegian University of Science and Technology (NTNU)  
Sem Sælands vei 7-9, N-7491 Trondheim, Norway  
Phone: +47 735 98716, Fax: +47 735 94466

## **Abstract**

The concept *code mobility* is used to describe the capability of code to dynamically change the bindings between code fragments and the location where they are executed. This paper summarizes the concept of code mobility by illustrating the related technologies, paradigms and application domains. It also introduces a framework of concepts, models, and terms to understand, assess and compare the different facets of the mobile code approach.

Key words: Mobile code, Mobile agent, distributed application, design paradigm

## **1. Introduction**

The increase in size and performance of computer network is both the cause and the effect of an important phenomenon: networks are becoming pervasive and ubiquitous. By pervasive, we mean that network connectivity is no longer an expensive add-on, rather it is often implicitly assumed when considering a computing platform. By ubiquitous, we refer to the availability of network connectivity independently of the physical location. Development in wireless technology free network hosts from the constraint of being placed at a fixed physical location and enable a new scenario where hosts, still connected to the net by wireless links, follow their users while their physical location is changing.

At the same time, the availability of distributed applications and systems accessible to the general public has triggered the interest of end users and markets, motivating the development of entirely new classes of application. Nonetheless, the range of applications capable of fully exploiting the underlying communication infrastructure is still relatively limited. This is mainly due to the lack of suitable mechanisms and abstractions that would allow developers to take full advantage of the potential of the new communication infrastructure. In addition, the new services and modes of use of computer networks often demand for a higher degree of customizability by the end user, whereas the dynamic nature of both the underlying communication infrastructure and the market requirement demands for increased flexibility and extensibility.

There have been many attempts to design a computational infrastructure that can match the characteristics of the communication infrastructure. These efforts attack different facets of the problem at different layers of abstraction. Most of the approaches, however, try to adapt well-established models and technology within the new setting, and usually take for granted the traditional client-server architecture. For example, CORBA[1] integrates remote procedure calls (RPC) with the object-oriented paradigm. It attempts to combine the benefits of the latter in terms of modularity and reuse with the well-established communication mechanism of the former. However, extending existing client-server technology is only a partial solution because it addresses just scaling and connectivity problems without providing new mechanisms and abstractions that allow the developers to structure the computation in a flexible, customizable and extensible way. A different approach comes from a promising research area, which is attacking the

problems mentioned by exploiting mobile code. Code mobility can be defined informally as the capability to dynamically change the bindings between code fragments and the location where they are executed [2].

Mobile code is an important programming paradigm and opens up new possibilities for structuring distributed software systems in an open and dynamically changing environment. It can improve speed, flexibility, structure, or ability to handle disconnections and it is particularly well suited if adaptability and flexibility are among the main application requirements. It has applications in many areas, such as mobile computing, active networks, network management, resource discovery, software dissemination and configuration, electronic commerce, and information harvesting [3].

A more elaborate form of mobile code, based on the "push principle" as opposed to the "pull principle" of mere code downloading, are mobile agents [3]. Mobile agents are programs that can move through a network under their own control, migrating from host to host and interacting with other agents and resources on each [4]. In contrast to simple mobile code systems, mobile agents have navigational autonomy; they decide on their own (based on their programmed strategy and the current state of the context) whether and when they want to migrate. While roaming the Internet or a proprietary intranet and visiting other machines, they do some useful work on behalf of their owners or originators.

This paper is structured as follows. Section 2 describes the motivation and approach of mobile code. Section 3 describes the current technologies that support the code mobility. Section 4 introduces some typical mobile code paradigms. Section 5 overviews some application domains of mobile code. Section 6 draws a conclusion.

## **2. Motivation and approach**

### **2.1 Related work**

Code mobility is not a new concept. In the recent past, several mechanisms and facilities have been designed and implemented to support the migration of code among the nodes of a network, such as process migration and object migration.

Process migration mechanisms manage the bindings between the process and its execution environment to allow the process to resume its execution seamlessly in the remote environment. Process migration facilities have been introduced the operation system level to achieve load balancing across network nodes. Therefore, most of these facilities provide transparent process migration, i.e. the programmer has neither control nor visibility of migration.

Object migration makes it possible to move objects among address spaces, implementing a finer grained mobility with respect to systems providing migration at the process level. An example of system providing transparent migration is the COOL [5] object-oriented subsystem.

### **2.2 The mobile code approach**

Process and object migration address the issues that arise when code and state are moved among the hosts of a loosely coupled, small-scale distributed system. However, they are insufficient when applied in larger scale settings. Nevertheless, the migration techniques discussed so far have been taken as a starting point for the development of a new breed of systems providing enhanced forms of code mobility. These systems often referred to as *Mobile Code Systems* (MCSs) [6].

## **2.3 A mobile code framework**

Many MCSs have been proposed. This lively and sometimes chaotic scenario has generated some confusion about concepts, abstractions, terms, and semantics of code mobility. Therefore, a conceptual framework is needed to foster understanding of the multi-faceted mobile code scenario and enable researchers and practitioners to assess and compare different solutions with respect to a common set of reference concepts and abstractions-and go beyond it. For this reason, classification is provided in this paper. The classification introduces abstractions, models, and terms to characterize the different approaches to code mobility proposed so far, highlighting commonalities, differences, and applicability. The classification is organized along three dimensions that are of paramount importance during the actual development process: technologies, design paradigms, and application domains. Mobile code technologies are the languages and systems that provide mechanisms enabling and supporting code mobility. The application developer uses mobile code technologies in the implementation stage. Design paradigms are the architectural styles that the application designer uses as building blocks in defining the application architecture. An architectural style identifies a specific configuration for the components and their mutual interactions. Application domains are classes of applications that share the same general goal, e.g. distributed information retrieval or electronic commerce. They play a role in defining the application requirement. The expected benefits of code mobility in a number of application domains are the motivating force behind this research field.

## **3. Mobile Code Technologies**

### **3.1 Introduction**

The interest in code mobility has been raised by the availability of a new breed of technologies featuring the ability to move portions of application code and possibly the corresponding state among the nodes of a wide-area network. These technologies apparently provide very diverse abstractions and primitives. In this chapter we analyze, classify, and compare different mechanisms provide by existing technologies, which are constituted mainly by programming languages and corresponding run-time support.

### **3.2 A distributed virtual machine**

Traditional distributed systems can be accommodated in the virtual machine shown on the following Figure 1.

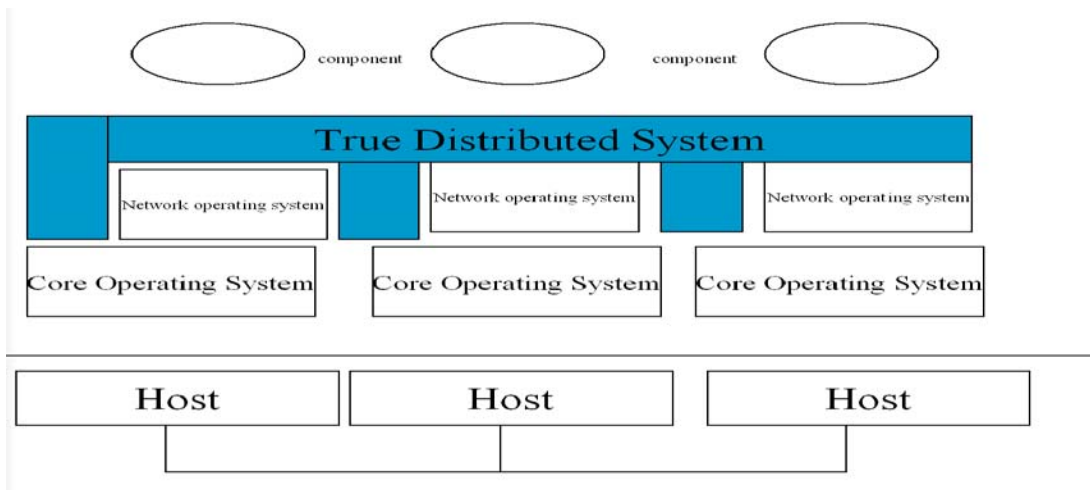


Fig 1. Traditional systems

The lowest layer, just upon the hardware, is constituted by *Core Operating System (COS)*. The COS can be regarded as the layer providing the basic operating system functionalities, such as file system, memory management, and process support. This layer provides no support for communication or distribution. The *Network Operating System (NOS)* layer provides the Non-transparent communication services. Application using NOS services address the host targeted by communication explicitly. For example, socket services can be regarded as belonging to the NOS layer, since a socket must be opened by specifying a destination network node explicitly. The NOS, at least conceptually, uses the services provided by the COS, e.g. memory management. Research on distributed systems has traditionally focused on achieving network transparency via a *True Distributed Systems (TDS)* layer. A TDS implements a platform where components, located at different sites of a network, are perceived as local. Users of TDS services do not need to be aware of the underlying structure of the network. When a service is invoked, there is no clue about the node of the network that will actually provide the service, and even about the presence of a network at all. As an example, CORBA services can be regarded as TDS services since a CORBA programmer is usually unaware of the network topology and interacts with a single, well-known object broker. At least in principle, the TDS is built upon the services provided by the underlying NOS.

Technologies supporting code mobility take a different perspective. The structure of the underlying computer network is not hidden from the programmer rather it is made manifest. In the following Figure 2, the TDS is replaced by *Computational Environments (CEs)* layered upon the NOS of each network host. In contrast with the TDS, the CE retains the “identity” of the host where it is located. The purpose of the CE is to provide applications with the capability to dynamically relocate their components on different hosts. Hence, it leverages off of the communication channels managed by the NOS and of the low-level resource access provided by the COS to handle the relocation of code, and possibly of state, of the hosted software components.

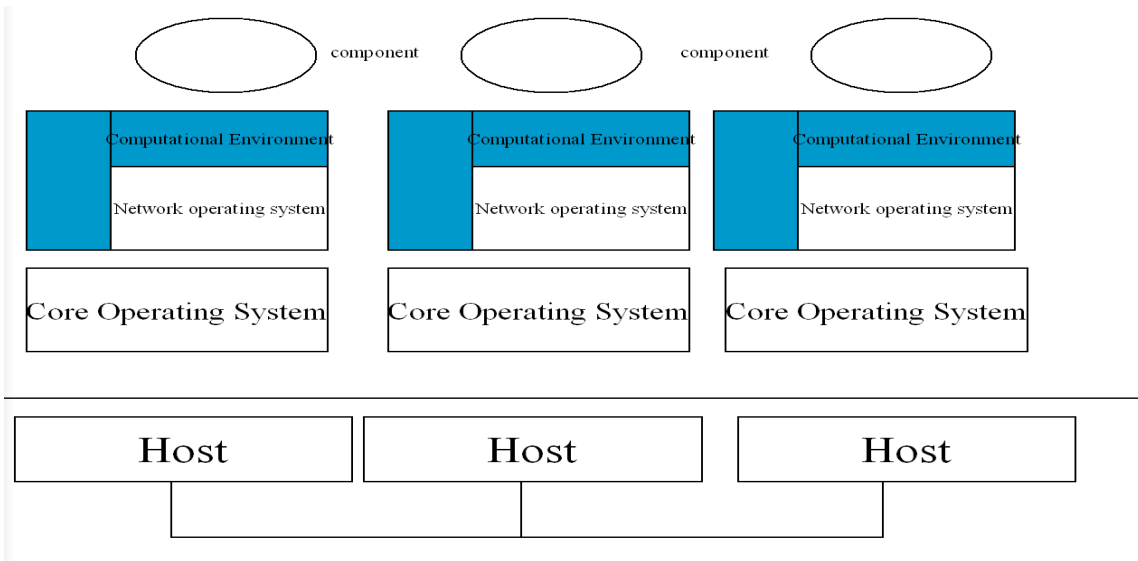


Fig 2. Mobile Code System

To give uniformity to our treatment, we distinguish the components hosted by the CE in executing units (EUs) and resources. Executing units represent sequential flow of computation. Typical examples of EUs are single-threaded processes or individual threads of a multi-threaded process. Resources represent entities, which can be shared among multiple EUs, such as a file in a file system, an object shared by threads in a multi-threaded object-oriented language, or an operating system variable. The following Figure 3 provides the static description for the composition of a *code segment*, which provides the static description for the behavior of a computation, and a *state* composed of a *data space* and an *execution state*. The data space is the set of references to components that can be accessed by the EU. As it will be explained later, these components are not necessary co-located with the EU on the same CE. The execution state contains private data that cannot be shared, as well as control information related to the EU state, such as the call stack and the instruction pointer.

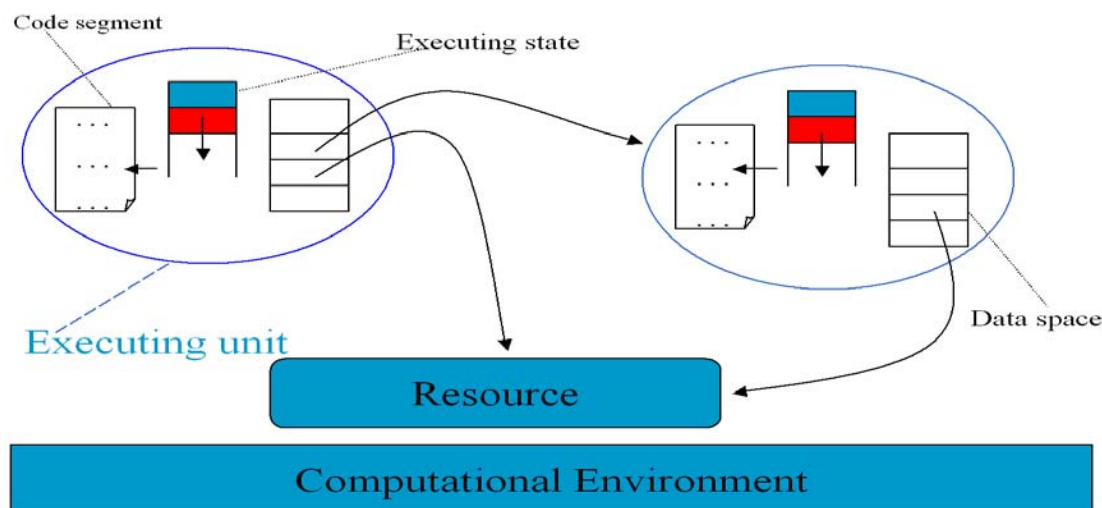


Fig 3. The internal structure of an executing unit

## 3.3 Mobility mechanisms

### Code and Execution State Mobility

Existing MCSs offer two forms of mobility, characterized by the EU constituents that can be migrated. *Strong mobility* is the ability of an MCS to allow migration of both the code and the execution state of a EU to a different CE. Weak mobility is the ability of an MCS to allow code transfer across different CEs; code may be accompanied by some initialization data, but no migration of execution state is involved. Strong mobility is supported by two mechanisms: migration and remote cloning. The migration mechanism suspends a EU, transmits it to the destination CE, and then resumes it. Migration can either be proactive or reactive. In proactive migration, the migrating EU determines the time and destination for migration autonomously. In reactive migration, a different EU that has some kind of relationship with the EU to be migrated triggers movement, e.g., an EU acting as a manager of roaming EUs. The remote cloning mechanism creates a copy of a EU at a remote CE. It differs from the migration mechanism because the original EU is not detached from its current CE. As in migration, remote cloning can be either proactive or reactive.

Mechanisms supporting weak mobility provide the capability to transfer code across CEs and either link it dynamically to a running EU or use it as the code segment for a new EU. Such mechanisms can be classified according to the direction of code transfer, the nature of the code being moved, the synchronization involved, and the time when code is actually executed at the destination site. As for direction of code transfer, a EU can either fetch the code to be dynamically linked and/or executed, or ship such code to another CE. The code can be migrated either as stand-alone code or as a code fragment. Stand-alone code is self-contained and will be used to instantiate a new EU on the destination site. Conversely, a code fragment must be linked in the context of already running and eventually executed. Mechanisms supporting weak mobility can be either synchronous or asynchronous, depending on whether the EU requesting the transfer suspends or not until the code is executed. In asynchronous mechanisms actual execution of the code transferred may take place either in an immediate or deferred fashion. In the first case, the code is executed as soon as it is received, while in a deferred scheme execution is performed only when a given condition is satisfied e.g., upon first invocation of a portion of the code fragment or as a consequence of an application event.

### Data Space Management

Upon migration of a EU to a new CE, its data space, i.e. the set of bindings to components accessible by the EU, must be rearranged. This may involve voiding bindings to resources, reestablishing new bindings, or even migrating some resources to the destination CE along with the EU. The choice depends on the nature of the resource involved, the type of binding to such resources, as well as on the requirement posed by the application.

We model resources as a triple  $\text{Resource} = \langle I, V, T \rangle$ , where  $I$  is a unique identifier,  $V$  is the value of the resources, and  $T$  is its type, which determines the structure of the information contained in the resources as well as its interface. The type of the resource determines also whether the resource is transferable or not transferable, i.e. whether, in principle, it can be migrated over the network or not. For example, a resource of type “stock data” is likely to be transferable, while a resource of type “pointer” probably is not.

Resources can be bound to a EU through three forms of binding, which constrain the data space management mechanisms that can be exploited upon migration. The strongest form of binding is by identifier. In this case, the EU requires that, at any moment, it must be bound to a given uniquely identified resource. A binding established by value declares that, at any moment, the resource must be compliant with a given type and its value cannot change as a consequence of migration. The weakest form of binding is by type. In this case, the EU requires that, at any moment, the bound resource is compliant with a given type, no matter what its actual value or identity are. This kind of binding is exploited typically to bind resources that are available to every CE, like system variables, libraries, or network devices.

## Execution and translation mechanisms

The choice of executing a program either through direct interpretation or through compilation is not distinctive of MCSs. Nonetheless; the nature of MCSs introduces new requirements as well as new degrees of freedom, which affect the criteria determining the choice. Code migrated among MCSs should be:

**Portable** The target platform is a computer network made of heterogeneous architectures. The obvious goal is to write the mobile code once, and then be able to unleash it for execution on any machine, without being aware of the specific hardware and software requirements.

**Secure** Code being executed within a CE must be checked, in order to protect the system from malicious or accidental damage.

As for security, interpretation enables load-time or run-time checks on the source code in order to verify that only legal instructions are executed. A compiled approach would force the run-time support of the sender machine to be aware of the platform of the receiver machine in order to select the appropriate native executable code for transmission. A common strategy among MCSs is the adoption of a hybrid approach in which source programs are compiled into an intermediate language that is then interpreted on the target machine. Using this technique, some MCSs push even further the portability issue, by achieving language independence in addition to platform independence. In this setting, CEs may support the execution of EUs whose code is written using different languages. Each CE manages several *virtual machines* each specialized for the execution of programs written in a particular language. A MCSs featuring more than one virtual machine, called a multi-language MCS, is a generalization of conventional MCSs where the language is fixed, e.g., Java or Telescript.

## Communication mechanisms

Mechanisms supported by existing MCSs can be distinguished along two orthogonal dimensions: the number of EUs involved in communication, and their mutual location. This first dimension partitions the space of mechanisms in *point-to-point* and *multi-point* communication mechanisms. The second dimension distinguishes among local and remote mechanisms, depending on whether the EUs are co-located in the same CE or not.

### Point-to-point mechanisms

Point-to-point mechanisms enable communication between two EUs. The primitive mechanism is message passing, e.g., remote procedure call. This may involve invocation of bare procedures or method invocation on remote objects, e.g., CORBA. RPC [7] is often exploited for both local and remote communication, in order to provide a uniform mechanism. *Streams* are different communication mechanisms that allow one EU to open a channel to another EU, and send it a continuous stream of data.

### Multi-point mechanisms

Multi-point mechanisms enable communication between more than two EUs at a time. *Shared memory* is by far the mechanism most frequently used to achieve multi-point communication. In order to communicate, several EUs are given a reference to the same variable or object. Changes in the value associated with the object are perceived by all the EUs owing a reference to it. *Event-based* mechanisms, in turn, define an event bus constituting the logical channel through which events are dispatched, together with primitives that allow EUs to generate events and to subscribe for receiving events they are interested in. Another form of implicit communication is provided by triple spaces. With this mechanism, EUs communicate by either inserting the triples containing the information to be communicated into a shared triple space, or by searching it for a triple using some form of pattern matching.

## Security mechanisms

Security is a big concern in systems providing support to code mobility and is often considered to be the main limitation to the wide acceptance of mobile code technologies [8]. MCSs provide a distributed computing infrastructure on which application belonging to different users can execute concurrently. In addition, the CEs that compose the infrastructure may be managed by different authorities with different and possibly conflicting objectives and may communicate across untrusting communication infrastructures, e.g., the Internet.

The security issues could be divided into two classes: intra-CE security, i.e., the issues related to the problem of protecting entities inside a CE, and inter-CE security that include the issues raised by interaction among remote entities.

### **Inter-CE security**

Inter-CE security issues encompass authentication, integrity, and privacy of the communication between two CEs, a EU and a remote CE, and two remote EUs. When interacting, CEs need to authenticate each other in order to be protected from spoofing. Authentication mechanisms can be based simply on identifiers that provide little or no security or may use some stronger mechanism based on cryptography.

### **Intra-CE security**

Intra-CE security addresses the problem of protecting the entities that compose a single computation environment. Intra-CE security encompasses security among different EUs, protection of the hosting CE and the associated resources from EUs, and protection of EUs from the CE. Most of these issues are addressed by authorization that determines the acceptable action of an entity on the basis of its identity, as determined by the authentication process.

## **4. Mobile Code Paradigms**

Traditional approaches to software design are not sufficient when designing large scale distributed application that exploit code mobility and dynamic reconfiguration of software components. In this case, the concept of location, distribution of components among locations, and migration of component to different location need to be taken explicitly into account during the design stage.

### **4.1 Basic concepts**

*Components* are the constituents of the software architecture. They can be further divided into *code components*, that encapsulate the “know-how” to perform a particular computation, *resource components*, that represent data or devices used during the computation, and *computational components*, that are active executor capable to carry out a computation, as specified by a corresponding know-how. *Interactions* are events that involve two or more components, e.g., a message exchanged between two computational components. *Site* host components and support the execution of computational components. A site represents the intuitive notion of location. Interactions among components residing at the same site are considered less expensive than interactions taking place among components located in different sites. In addition, a computation can be actually carried out only when the know-how describing the computation, the resources used during the computation, and the computational component responsible for execution are located at the same site [6].

We identify four main design paradigms exploiting code mobility: remote evaluation, push, code on demand, and mobile agent. These paradigms are characterized by the location of components before and after the execution of the service, by the computational component, which is responsible for execution of code, and by the location where the computation of the service actually takes place.

## 4.2 Paradigms

In the *Remote Evaluation (REV)* paradigm [9], a computational component is transferred from the Client to the Server, which executes the code using its resources and delivers the results back to the Client. Figure 4 shows that the REV Interacting process consists of Code Sending, which transfers the Requested Processing from the Client to the Server, Code Activating, which invokes Requested Processing on the Server, and Result Retrieving, which transfers the Requested Result when ready. As the event link from the Client's Activation Request shows, the Client initiates the REV Interacting process.

In the *Code On Demand (COD)* paradigm [2], the Client is able to access the resource components it needs, but has no knowledge about how to process such resource components. Thus, the Client interacts with the Server, which hosts the Requested Processing. Contrarily to the REV paradigm, in the COD paradigm the Requested Processing is executed in the Client. Figure 5 shows that the COD Interacting process consists of Code Retrieving, which transfers the Requested Processing from the Server to the Client, and Code Activating, which invokes Requested Processing in the Client. This invocation might change the Required Data and produces the Requested Result. Here too, the Client initiates the COD Interacting process.

In the *PUSH* paradigm [10], as opposed to the REV and COD paradigms, the Server sends a (computational or resource) component to the Client in advance of any specific request. This push-based operation is often preceded by a profiling operation, in which the Client specifies a profile, which reflects its user's interests. This profile is sent to the Server, saved there, and used to decide what components the Client should receive and when to send them. The advantage of this paradigm is that the users do not have to know when to pull new components and where to pull them. Rather, the system sends new components automatically when necessary. PUSH Interacting, shown in Figure 6, is triggered due to a change in the Requested Component state, only if the Profile is set to client (as the event and condition links show, respectively). When activated, PUSH Interacting first transfers the Requested Processing from the Server to the Client and then activates it (in the Client).

In the *Mobile Agent (MA)* paradigm [11], the agent starts executing in the Server. Since some of the Required Data is located in the Client, the agent migrates to the Client and completes its execution using the resource components available there. The agent itself usually initiates the migration. Contrary to the previous paradigms, which focus on the transfer of just code between sites, the mobile agent migrates to the remote site as a whole computational component, along with its state, the code it needs, and some of the resource components required to perform the task. In Figure 7, the agent, Requested Processing, which is characterized by Private Data and Execution Status, is hosted and executed in the Server. When the agent's Execution Status enters its transfer state, the MA Interacting process is activated (as denoted by the event link), cloning the agent to the Client, activating it, and changing its state to local. Then the agent can continue its execution in the Client, using additional Required Data and producing the Requested Result.

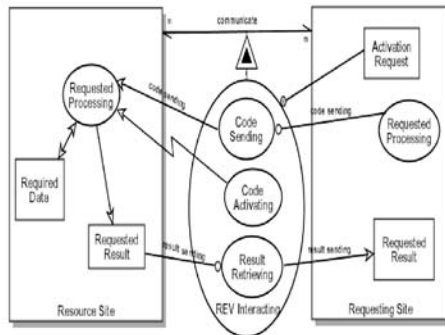


Fig. 4 REV

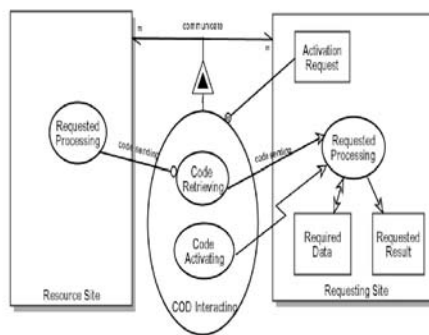


Fig. 5 COD

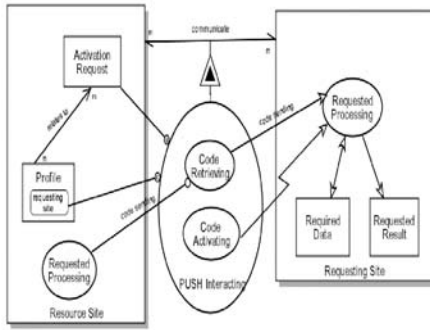


Fig. 6 PUSH

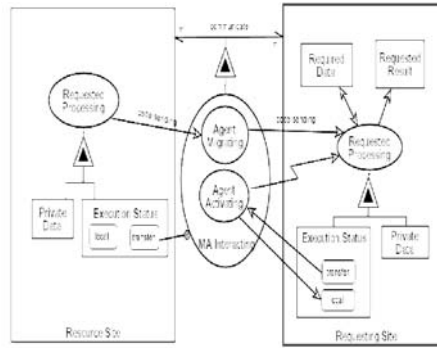


Fig. 7 MA

## 5. Mobile Code Applications

This chapter introduces some application domains, which are being identified by researchers in the field as suitable for the exploitation of mobile code.

### 5.1 Key benefits of mobile code

A major asset provided by code mobility is that it enables *service customization*. In conventional distributed systems built following the CS paradigm, servers provide some set of services accessible through a statically defined interface. It is often the case that this set of services, or their interfaces, are not suitable for unforeseen client needs. A common solution to this problem is to upgrade the server with new functionality, thus increasing both its complexity and its size without increasing its flexibility. The ability to request the remote execution of code, by converse, helps in increasing server flexibility without affecting permanently the size or complexity of the server. In this case, in fact, the server actually provides very simple and low-level services that seldom need to be changed. The client to obtain a customized high-level functionality that meets the specific client's needs then composes these services.

### 5.2 Application domains for mobile code

#### Distributed information retrieval

The study of distributed information retrieval lies at the intersection between information retrieval and distributed systems. From the former comes the goal of effectiveness, that a person who enters a query in the broker should find relevant information [12]. The information sources to be visited can be defined statically or determined dynamically during the retrieval process. This is a wide application domain, encompassing very diverse applications. Code mobility could improve efficiency by migrating the code that performs the search process close to the information base to be analyzed.

#### Active documents

In active documents applications, traditionally passive data like e-mail or web pages, are enhanced with the capability of executing programs which are somewhat related with the document contents, enabling enhanced presentation and interaction. Code mobility concepts are fundamental of these applications since they enable the embedding of code and state into documents, and support the execution of the dynamic contents during document fruition. A typical choice is a combination of WWW technology and Java

applets. Java applets are interactive applications that can be dynamically pulled across the network with Java-enabled WWW browser [13].

### **Advanced telecommunication services**

Support, management, and accounting of advanced telecommunication services like videoconference, video on demand, or telemeeting, demand for a specialized middleware providing mechanisms for dynamic reconfiguration and user customization. For example, the application components managing the setup, a service broker could dispatch signaling, and presentation services for a videoconference to the user. A particular class of advanced telecommunications services is those supporting mobile users. In this case, autonomous components can provide support for disconnected operations.

### **Remote device control and configuration**

Remote device control applications are aimed at configuring a network of devices and monitoring their status. This domain encompasses several other application domains, e.g., industrial process control and network management. In the classical approach, monitoring is achieved by periodical polling of the resource state and configuration is performed using a predefined set of services. This approach, based on the CS paradigm, can lead to a number of problems. Code mobility could be used to design and implement monitoring that are co-located with the devices being monitored and report events that represent the evolution of the device state. In addition, the shipment of management components to remote sites could improve both performance and flexibility [14][15].

### **Workflow management and cooperation**

Workflow management deals with the specification and execution of business processes. General workflow specifications include the actions to be performed, routing information and policies that describe the organizational environment [16].

The workflow defines which activities must be carried out in order to accomplish a given task as well as how, where, and when these activities involve each party. A way to model this is to represent activities as autonomous entities that, during their evolution, are circulated among the entities involved in the workflow. Code mobility could be used to provide support for mobility of activities that encapsulate their definition and state. For example, a mobile component could encapsulate a text document that undergoes several revisions. The component maintains information about the document state, the legal operations on its contents, and the next scheduled step in the revision process.

### **Electronic commerce**

E-commerce has grown at an astounding rate over the last few years and so has the number of participants. Although web business models differ from the traditional brick and mortar models in many ways, the fundamental needs of consumers and businesses remain much the same: consumers want to compare shop products and services side by side for the best price; businesses want to grow sales by driving the right shoppers to their sites. These needs give rise to a variety of intelligent agents working for buyers and sellers of products and services over the web [17].

Certainly, electronic commerce is one of the most attractive areas in that respect: a mobile agent may act (on behalf of a user or owner) as a seller, buyer, or trader of goods, services, and information. Accordingly, mobile agents may go on a shopping tour in the Internet – they may locate the best or cheapest offerings on WWW servers, and when equipped with a negotiation strategy (i.e., if they are "intelligent agents") they may even do business transactions on behalf of their owners [18].

## **6. Conclusion**

Mobile code is a promising solution for the design and implementation of large-scale distributed applications, since it overcomes many of the drawbacks of the traditional client-server approach.

In this paper, we introduced a conceptual framework structured along three classes of concepts: technologies, design paradigms, and applications. Technologies support application development. Paradigms guide the design of applications. Applications are the solutions to specific problems. The purpose of the framework presented in this work is to foster progress towards a common understanding of the issues and contributions in the area of code mobility.

## **References:**

- [1] Object Management Group, "CORBA: Architecture and Specification." Aug. 1995
- [2] A. Carzaniga, G.P. Picco, and G. Vigna, "Designing Distributed Applications with Mobile Code Paradigms," Proceedings of 19<sup>th</sup> International Conference on Software Engineering, pp 22-32. ACM press, 1997
- [3] D. Kotz, F. Mattern (Eds.), "Agent Systems, Mobile Agents, and Applications," Springer-Verlag, 2000
- [4] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, George Cybenko, " Mobile agents for mobile computing," Technical Report PCS-TR96-285, 1996, [ftp://ftp.cs.dartmouth.edu/TR/TR96-285.pdf](http://ftp.cs.dartmouth.edu/TR/TR96-285.pdf)
- [5] R. Lea, C. Jacquemont, and E. Pillevesse, "COOL: System Support for Distributed Object-Oriented Programming," Communications of the ACM, vol. 36, no. 9, pp 37-46. Nov. 1993
- [6] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, "Understanding Code Mobility," IEEE transaction on Software Engineering, vol. 24, no.5, pp 342-361. May. 1998
- [7] A. Birrell and B. Nelson, "Implementing Remote Procedure Calls. ACM transactions on Computer Systems," 2(1), pp 29-59. Feb. 1984
- [8] G. Vigna. Editor, "Mobile Agents and Security." LNCS. Springer, 1998
- [9] Stamos, J. and G. Gifford, "Remote Evaluation." ACM Transactions on Programming Languages and Systems, vol.12, no. 4, pp 537-565. 1990
- [10] Franklin, M. and S. Zdonik, "Data In Your Face: Push Technology in Perspective." ACM SIGMOD International Conference on Management of Data, pp. 516-519. 1998
- [11] Gray, R., D. Kotz, G. Cybenko, and D. Rus, "Mobile Agent: Motivations and State of- the-art Systems." In Bradshaw J. M. (Ed.), Handbook of Agent Technology, AAAI/MIT Press. 2000.
- [12] Nicholas Eric Craswell, "Methods for Distributed Information Retrieval," <http://pigfish.vic.cmis.csiro.au/~nickc/pubs/craswellthesis.pdf>.
- [13] "The Java Language: A white paper," Sun Microsystems White Paper, Sun Microsystems, 1994
- [14] M. Baldi, S. Gai, and G.P. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," Proceedings of the First International Workshop on Mobile Agents, pp. 13-16. 1997
- [15] G. Goldszmidt and Y. Yemini, "Distributed Management by Delegation," Proceedings of 15<sup>th</sup> International Conference on Distributed Computing, Jun. 1995
- [16] Ting Cai, Peter Gloor, and Saurab Nog, " Dataflow: A workflow management system on the Web using transportable agents," Technical Report TR96-283, 1996

[17] M.V.Sinmao, "Intelligent Agents and E-Commerce",  
<http://www.msb.edu/faculty/culnanm/EC/Briefings2/Sinmaom.htm>

[18] S. Fünfroeken, F. Mattern, "Mobile Agents as an Architectural Concept for Internet based Distributed Applications," The WASP Project Approach, in: Steinmetz (ed.) Proceedings of KIVS'99, Springer-Verlag, 1999