Abstract

Web applications exhibit dynamic behavior through such features as animation, rapidly changing presentations, and interactive forms. The growing complexity of Web applications requires a rigorous modeling approach that would be capable of clearly and explicitly addressing code mobility issues. While mobile agent systems and programming languages support the implementation of code mobility with features such as applets or mobile agents, existing system analysis and design methods lack the facilities to model code mobility at a satisfactory level. OPM/Web is an extension of Object-Process Methodology (OPM) for modeling distributed systems and Web applications that enables intuitive modeling of code mobility concepts in a single framework. In this paper, we propose generic OPM/Web models for common code mobility design paradigms, including Remote Evaluation, Code-on-Demand, PUSH, and Mobile Agents. An OPM/Web model of a mobile application that handles requests for Quality of Service over the Internet exemplifies the use and advantages of modeling such systems in OPM/Web.

Keywords: mobile code, code migration, code mobility design paradigms, Web application modeling, Object-Process Methodology.

1. Introduction

Although Web applications seem to exhibit a relatively simple distributed architecture, the underlying architecture is dynamic and complex. The complexity arises from the requirements
Web applications to respond to an unlimited number of heterogeneously skilled users, address security and privacy concerns, access heterogeneous, up-to-date information sources, and exhibit dynamic behavior. The growing complexity of Web applications requires a rigorous modeling approach. Such approach should be capable, among other things, of addressing code mobility issues to enable dynamic reconfiguration of the binding between software components and their physical locations. **Code mobility** is the capability of software systems to dynamically reconfigure the binding between the software components of an application and their physical locations (nodes) within a computer network (Fugetta et. al., 1998). **Mobile Code** is a piece of code that exhibits the mobility property, i.e., code that can be transmitted across a network and executed on another node. **Code migration** is the function which controls how code mobility is achieved (Dale and DeRoure, 1997). Although most applications do not require mobile code, adding this capability to applications supports disconnected operations and can enhance system flexibility, reduce bandwidth consumption and total completion time, and improve fault tolerance (Fugetta et. al., 1998).

The code migration process involves determining the operation targets, transferring the code, and integrating it into the target system. In static system architectures, the targets can be determined at compilation time. If the system architecture is dynamic, the operation targets should be computed immediately prior to transferring the code. Following the target determination, the code can be transferred by applying one of the design paradigms for code mobility, which extend the traditional client-server paradigm from data to code. Once transferred, the code can be integrated with the local target system by activating an instance of it, connecting it to existing data or code, or continuing its transfer over the network to yet another target.
Modeling the code migration process also includes defining process triggers, preconditions and postconditions, and handling security issues and possible transfer errors.

Current techniques for modeling code mobility and migration require determining the operation targets separately from the transferring stage (e.g., by class services) and do not specify how the code is to migrate. In the object-oriented approach, the description of code migration is scattered. For example, in UML (Object Management Group, 1999), which is the standard object-oriented modeling language, code migration specifications are decomposed into at least five views: use case, class, interaction, state, and deployment diagrams. This decomposition is hard to integrate to a whole, consistent system and is also complicated to maintain (Mezini and Lieberherr, 1998).

OPM/Web, which is the extension of Object-Process Methodology (OPM) to distributed systems and Web applications, constitutes a complete approach to modeling the structure and behavior of a system within a single view by considering objects and processes as two equally important classes of entities. The purpose of this paper is to show how OPM/Web can clearly model all the important aspects of code migration. In Section 2, we review the literature concerning the main concepts and design paradigms of code mobility, and discuss the shortcomings of existing modeling techniques in specifying code migration. In Section 3, we connect and map OPM/Web concepts to the terminology of mobility, while in Section 4 the main code mobility design paradigms are modeled in OPM/Web. Section 5 explains and demonstrates how to use these models in a complete mobile application, which handles requests for Quality Service. Section 6 summarizes and discusses OPM/Web advantages and shortcomings in modeling code migration.
2. **Modeling Code Mobility: Literature Review**

Applications that involve code mobility are defined in terms of components, interactions, and sites (Carzaniga et al., 1997). *Components* are the building blocks of system architecture. They are further divided into *resource components*, which are objects (architectural elements representing data, or physical devices), and *computational components*, which are programs that embody flows of control. A resource component is represented in object-oriented terms as an object with attributes and operations (services) that contain knowledge about how to execute a particular task, while a computational component, which contains code, may also be characterized by private data, an execution state, and bindings to other (resource or computational) components.

*Interactions* are events that involve two or more components communicating with each other. *Sites* are nodes or execution environments— they host components and provide support for the execution of computational components.

### 2.1 The Client-Server Paradigm and Related Approaches

The *Client-Server* (CS) paradigm (Renaud, 1993) is the traditional design approach for distributed communication among sites, in which messages are transferred from one site to another, but actual code is not. In a typical client-server interaction, site $S_B$, which acts as the interaction server, offers a set of services. It also hosts the resources and the knowledge needed for executing these services. Site $S_A$, which is the operation client, requests the execution of a service offered by $S_B$ by sending it a message. As a response, $S_B$ performs the requested service and delivers the result back to $S_A$ in a subsequent interaction. If the server does not have all the data and knowledge required, it can act as a client in another client-server interaction.

The CS paradigm has been criticized as being too low-level, requiring developers to determine network addresses and synchronization points. CS interaction is also too specific, since the cli-
must “know” the exact services that the server can provide (Dale and DeRoure, 1997). The Remote Procedure Call (RPC) (Bloomer, 1992) tries to overcome these shortcomings by permitting the client to request a service to be executed on a server in the same way a local function call is made; The location of the server, the initiation of the service, and the transportation of the results are handled transparently to the client. The object-oriented approach attempts to make the paradigm more accessible and uniform by adopting reuse, inheritance, and encapsulation principles. OMG’s Common Object Request Broker Architecture (CORBA) (Object Management Group, 1995) is a CS technology that is based on the object-oriented approach.

2.2 **Design Paradigms for Code Mobility**

Design paradigms for code mobility extend the CS paradigm by transporting computational components across a network. Four common design paradigms for code mobility are Remote Evaluation (REV), Code-on-Demand (COD), PUSH, and Mobile Agents (MA). These paradigms differ in their preconditions, postconditions, and triggers.

In the **Remote Evaluation** (REV) paradigm (Stamos and Gifford, 1990), a computational component, C, located at S_A, has the knowledge (represented by code) necessary to perform a service, but it lacks the required resource components, which are located at a remote site S_B. Therefore, C is transferred from S_A to S_B and is executed there. The results of this execution are delivered back to S_A in an additional interaction.

In the **Code-on-Demand** (COD) paradigm (Carzaniga et. al., 1997), site S_A can access the resource components needed for a service, but it does not have the knowledge required to process them. Therefore, S_A requests the service execution knowledge, i.e., the computation component C, from its hosting site, S_B. S_B delivers the knowledge to S_A, which subsequently processes C.
site $S_A$ on the resource components residing there. Contrary to REV, in COD the code is executed at the client.

In the **PUSH** paradigm (Franklin and Zdonik, 1998), site $S_B$ sends a (computational or resource) component to site $S_A$ in advance of any specific request. This push-based operation is often preceded by a profiling operation, in which $S_A$ specifies a profile that reflects its users’ interests. The profile is sent to site $S_B$, saved there, and used by $S_B$ to decide which components $S_A$ should receive and when to send them. The advantage of this paradigm over COD is that the users do not have to know when to pull new components and where to pull them from. Rather, the system automatically sends necessary new components when they become available, and they are often used later by the receiving node.

In the **Mobile Agent** (MA) paradigm (Gray et. al., 2000), site $S_B$ owns the service execution knowledge, $C$, but some of the required resource components are located at site $S_A$. Hence, $C$ migrates to $S_A$ and completes the service using the resource components available there. The migration is usually initiated by the agent ($C$), but it might be requested by $S_A$ or $S_B$. Contrary to the REV, COD, and PUSH paradigms, which focus on the transfer of just code between sites, a mobile agent migrates to the remote site as a whole computational component, along with state, the code it needs, and some of the resource components required to perform the task.

Discussing these design paradigms for code mobility, Carzaniga et al. (Carzaniga et. al., 1997) claim that none of them is absolutely better than the others and suggest choosing the most appropriate paradigm for a system under development on a case-by-case basis according to the application type and needs.
2.3 Modeling Code Mobility and Migration

Code mobility is supported by such programming environments as Java, Telescript (White, 199 and D’Agents (Gray et. al., 2001). However, current modeling techniques that are used in analysis and design phases of Web applications do not address code mobility concepts a satisfactory level.

Web applications can be classified as hybrids between hypermedia and information systems (Fraternali, 1999). Most commonly, such systems are modeled using hypermedia authoring techniques or visual software engineering methods, especially object-oriented ones. Hypermedia authoring techniques, including Hypertext Design Model (HDM) (Garzotto et. al., 1999), Relationship Management Methodology (RMM) (Isakowitz et. al., 1995), Object-Oriented Hypertext Design Model (OOHDM) (Schwabe and Rossi, 1998), and WebML (Ceri et. al., 2000) model the content and navigational aspects of an application, but not its functionality, physical architecture, or security requirements. Therefore, they do not explicitly address code-related issues, such as code migration.

Object-oriented development languages, notably UML (Object Management Group, 1999), enable modeling of the application functionality through class services and message passing among objects. Concepts involving code mobility, such as Java applets, are modeled in separate views using pre-declared UML stereotypes. Conallen’s extension of UML for Web applications (Conallen, 1999), for example, is based on a set of 18 domain-specific stereotypes, which are commonly used with Web applications. These stereotypes include such implementation dependent concepts as RMI, IIOP, and Java Script, along with a set of well-formedness rules for using them. In general, UML does not handle the code migration process as a whole pattern, including its preconditions (e.g., the existence of a request in the client site and source code at
server site), postconditions (e.g., the existence of executable code at the client site), and triggers (e.g., a change in a server component). To overcome these shortcomings, UML has been extended by various research teams, including the mobile agent extension (Klein et. al., 2000), Agent UML (AUML) (Odell et. al., 2000), and MASIF-DESIGN (Muscutariu and Gervais, 2001). Even though the proliferation of such extensions undermine and weaken UML standardization efforts, they still do not separate the execution knowledge (services) from the resource components (classes). It should come as no surprise that such separation is not possible, since doing so would work against the encapsulation of operations within object classes, which is a major principle in the object-oriented approach.

Behavior-oriented techniques, including Aspect-Oriented Design (AOSD site, 2003) and superimposition (Katz, 1993), model parts of the system functionality separately from the application structure. They enable static binding of processes to sites, but do not support modeling of dynamic configurations and the actual migration process.

Object-Process Methodology (OPM) (Dori, 2002) combines ideas from the object-oriented development methods and behavior-oriented techniques in order to specify the system structure and dynamics within a single framework. OPM enables the existence of processes as stand-alone entities. This way, structure and behavior, the two major aspects that each system exhibits, exist in the same OPM model without highlighting one at the cost of suppressing the other. Integrating structure and behavior, OPM provides a solid basis for modeling complex systems, which these two most prominent system aspects are highly intertwined and hard to separate. Mobile applications are prime examples of such systems. Since OPM lacks the ability to specify the code migration process and dynamic reconfiguration at run time, it has been extended OPM/Web, as discussed in the next section.
3. OPM/Web and Mobile Components

OPM/Web extends Object-Process Methodology to distributed systems and especially to Web applications, enabling the modeling of such systems within a single view. As in OPM, the OPM/Web universe of discourse is specified in terms of “things”: object classes and process classes. An object class (abbreviated as an object) is a set of object instances which exist, or at least have the potential of stable, unconditional physical or logical existence. A process class (abbreviated as a process) is a pattern of transformation of one or more object classes. A program, an operation, a procedure, and an algorithm are examples of process classes. An actual execution of a process (such as the carrying out of an executable version of a program or an algorithm) is a process instance. The relations between things (objects and processes) is modeled by structural links (e.g., generalization and aggregation) and procedural links (which specify transformations, enablers, and triggers). Contrary to object-oriented methods, an OPM process can stand alone and involve several object classes.

OPM enables managing the complexity of a model applying three refining/abstracting mechanisms: unfolding/folding, in which the thing being refined is shown as the root of a structural graph; in-zooming/out-zooming, in which the thing being refined is blown up to enclose its constituents; and state expression/suppression, which allows showing or hiding the possible states of an object. Using flexible combinations of these three scaling mechanisms, OPM enables specifying a system to any desired level of detail without losing legibility and comprehension of the resulting specification.

Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related Object-Process Diagrams (OPDs) constitute the graphical, visual OPM formalism. Each OPM element is denoted in an OPD by a symbol, and
OPD syntax specifies correct and consistent ways in which entities can be linked. The Object Process Language (OPL), defined by a grammar, is the textual counterpart of the graphical OPD set. OPL is a dual-purpose language, oriented towards humans as well as machines. Catering to human needs, OPL is designed as a constrained subset of English, which serves domain experts and system architects engaged in analyzing and designing a system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase. While the OPD set and the OPL script are equivalent in their semantic content, they are complementary from a human cognition viewpoint. Designed also for machine interpretation through a well-defined set of production rules, OPL provides a solid basis for automatically generating the designed application. An integrated software engineering environment, called OPCAT (Object-Process CASE Tool) (Dori et al., 2003), automatically translates from one modality to the other in either direction.

OPM/Web enhances the ability of OPM to model distributed systems in general and Web applications in particular in two ways. The first extension is the ability to reuse component designs in an open manner through bindings among model components, thereby improving model scalability (Reinhartz-Berger et al., 2002). The second extension is the support for code mobility and migration specifications. In this paper we focus on defining and modeling code mobility concepts and design paradigms using OPM/Web.

3.1 Mapping Mobility Terms onto OPM/Web Concepts

The terms used in the various design paradigms for code mobility are mapped to OPM/Web concepts as follows.

- A resource component is an informatical or physical object. An informatical object is a piece of information, such as the data required for a process execution. A physical object is tangible in the broad sense, for example a device.
A computational component is a process. It can own private data (objects) and include processes. The migration process can transfer the computational component source code (i.e., a process class), which can be compiled at the target site and run there any number of times, or an executable version of the code (i.e., a process instance), which can run at the target site only a specified number of times.

A site, which is analogous to a node in the UML implementation model, is a physical object in OPM/Web. This physical object can be in-zoomed to expose its resource and computational components.

An interaction has both structural and dynamic aspects. The structural aspect of an interaction specifies how two sites can communicate with each other, irrespective of a specific point in time. This aspect is modeled in OPM/Web by a (unidirectional or bi-directional) structural link between the communicating sites, which, as noted, are physical objects. The dynamic aspect of an interaction is the ability to transfer data (objects) or code (processes) between two sites and is specified in OPM/Web by an event-driven process. Since interaction conceptually characterizes the communication between the sites, the interaction process associated in the model to the structural link that connects the two interacting sites. Implementation of this interaction may still be carried out as two inter-related processes, one at each interacting site.

A summary of the main OPM/Web symbols and their meanings is provided in Appendix A. The basic code transferring operations are represented by the generic OPDs in Figure 1. The computational Component on the left of Figure 1(a) and Figure 1(b), which is a process class denoted by an ellipse, is the (unchangeable) input for the Component Transferring process, as the instrument link between them indicates. In Figure 1(a) the Component Transferring process...
transfers Component’s source code, while in Figure 1(b) Component Transferring transfer: process instance, i.e., only an executable version of Component. Following the UML notation classes and objects, a process instance is denoted in OPM by an ellipse within which the process class name is written as :ProcessClassName, where the identifier of the instance can optionally precede the colon.

The semantics of the arrow with the white (blank) arrowhead from Component Transferring the right appearance of Component is a result link\(^1\), which means that Component Transferring creates (a copy of) the process class Component, as in Figure 1(a), or an instance of it, as Figure 1(b). The identical path labels\(^2\) on the instrument and result links and the identical component names indicate that Component Transferring transfers Component as is rather than computing it from an input.

![Diagram](image)

Figure 1. A generic OPM/Web model of a Component Transferring process.

(a) Component Transferring transfers Component’s code, leaving the original Component intact.

(b) Component Transferring transfers an instance of Component, leaving the original Component intact.

---

\(^1\) In the original OPM, processes are not connected, and, hence, there is no difficulty to determine which is the processing entity. To remove the ambiguity arising from connecting two processes in OPM/Web via consumption or result links, a consumption link is denoted as a black-headed arrow from the consumed entity to the process, while the semantics of a white-headed arrow from a process to an entity remains a result link.

\(^2\) A path label in OPM is a label on a procedural link that removes the ambiguity arising from multiple incoming/outgoing procedural links. Here we use identical path labels on the incoming link to and outgoing link from the Component Transferring process to denote the transfer flow.
3.2 Modeling the Client-Server Paradigm using OPM/Web

Based on the mapping of code mobility terms onto OPM/Web concepts, an OPM/Web model of the traditional client-server paradigm, presented in Figure 2, consists of two equivalent modalities: graphical – the OPD in Figure 2(a), and textual – the OPL paragraph in Figure 2(b). The objective of this unique dual representation is to enhance the readability of the model for humans: engineering-oriented readers, who are familiar with OPM and its diagrammatic notation, can relate to the OPD, while domain experts, or those who are new to the OPM graphic notation, can refer to the OPL paragraph and learn the correspondence between each OPL sentence and its OPD construct counterpart. The OPL paragraphs also improve system documentation.

Examining Figure 2, one can see that Requesting Site (the client) and Processing Site (the server) are both physical objects (as denoted by shadowed rectangles). The computational component, Requested Processing, resides in the Processing Site, which also hosts the resource components required for that computation, Required Data and (later on) Requested Result. The two sites are connected via a bi-directional structural link, tagged communicate, which exhibits (i.e., is characterized by) the CS Interacting process. A change in (an instance of) Activation Request at the Requesting Site initiates the CS Interacting process, as the event link (the circle headed arrow with the letter 'e' inside it) between the two things shows. Following the transferring path, the first subprocess of the CS Interacting process, which is Result Requesting, transfers a copy of Activation Request to the Processing Site. As soon as this copy is placed at the Processing Site, it activates the Requested Processing, as the consumption event link (the black headed arrow with the letter 'e' next to it) denotes. This Requested Processing potentially affects the Required Data object and yields (produces) the Requested Result object.
(b) Requesting Site physical.

Requesting Site zooms into Activation Request and Requested Result.

Activation Request triggers CS Interacting.

Processing Site is physical.

Processing Site zooms into Activation Request, Required Data and Requested Result, as well as Requested Processing.

Activation Request triggers Requested Processing when its state changes.

Requested Processing consumes Activation Request of Processing Site.

Requested Processing affects Required Data.

Requested Processing yields Requested Result of Processing Site.

Many Requesting Sites and many Processing Sites communicate, and this relation exhibits CS Interacting.

CS Interacting zooms into Result Requesting and Result Retrieving.

Following path request transferring, Result Requesting consumes Activation Request of Requesting Site.

Following path request transferring, Result Requesting yields Activation Request of Processing Site.

Following path result transferring, Result Retrieving consumes Requested Result of Processing Site.

Following path result transferring, Result Retrieving yields Requested Result of Requesting Site.

Figure 2. An OPM/Web model of the Client-Server (CS) paradigm:

(a) The OPD (b) The corresponding OPL paragraph
The creation of **Requested Result** enables the second stage of the interaction, executed

**Result Retrieving.** Following the **result transferring** path, this process moves the local copy of the generated **Requested Result** from the **Processing Site** to the **Requesting Site**.

Table 1 summarizes the structure of **Requesting Site** and **Processing Site** before and after activation of a **CS Interacting** process. The dynamic aspect of the **CS Interacting** process can vividly simulated using OPM Case Tool (OPCAT), as explained in Appendix B.

Table 1. The resource and computational components in **Requesting Site** (the client) and **Processing Site** (the server) before and after an activation of **CS Interacting**

<table>
<thead>
<tr>
<th><strong>Design Paradigm</strong> (Process Name)</th>
<th><strong>Time</strong></th>
<th><strong>Requesting Site</strong></th>
<th><strong>Processing Site</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Server (CS Interacting)</td>
<td>Before</td>
<td>Activation Request</td>
<td>Requested Processing (code) Required Data</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>Requested Result</td>
<td>Requested Processing (code) Required Data</td>
</tr>
</tbody>
</table>

4. **OPM/Web Models of Code Mobility Design Paradigms**

OPM/Web enables precise modeling of the REV, COD, PUSH, and MA paradigms, which were explained informally in Section 2.2. In this section, we present generic OPM/Web models of these design paradigms. In all of these models, **Requesting Site** is the transaction client, and such, it obtains a copy of the **Requested Result** and keeps it at the end of the process. **Activation Request** is the trigger for the code transferring process. The **Resource Site** is the transaction server, i.e., it hosts the **Requested Processing** (as in COD, PUSH, and MA) or the **Required Data** (as in REV). The COD, PUSH, and MA models describe transferring a one-time executable version of code (i.e., a process instance) from the **Resource Site** to the **Requesting Site**, and executing it in the remote site. The REV model specifies a process that transfers an executable version of code from **Requesting Site** to **Resource Site** and executes it there. Replacing
process instance with a process class supports transfer of source code that can later instantiated, i.e., compiled and executed. The various code mobility models can become gene components in specifications of mobile applications, as explained and demonstrated in Section

Table 2. The resource and computational components in Requesting Site (the “client”) and Resource Site (the “server”) before and after an activation of the transfer processes in each one of the four code mobility design paradigms.

<table>
<thead>
<tr>
<th>Code Mobility Design Paradigm (Process Name)</th>
<th>Time</th>
<th>Requesting Site</th>
<th>Resource Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Evaluation (REV Interacting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before</td>
<td></td>
<td>Activation Request</td>
<td>Required Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requested Processing code</td>
<td></td>
</tr>
<tr>
<td>After</td>
<td></td>
<td>Requested Processing code</td>
<td>Required Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requested Processing instance</td>
<td></td>
</tr>
<tr>
<td>Code-on-Demand (COD Interacting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before</td>
<td></td>
<td>Activation Request</td>
<td>Requested Processing code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requested Data</td>
<td></td>
</tr>
<tr>
<td>After</td>
<td></td>
<td>Requested Data</td>
<td>Requested Processing code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requested Processing instance</td>
<td></td>
</tr>
<tr>
<td>PUSH (PUSH Interacting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before</td>
<td></td>
<td>Required Data</td>
<td>Requested Processing code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Profile</td>
<td>Activation Request</td>
</tr>
<tr>
<td>After</td>
<td></td>
<td>Required Data</td>
<td>Requested Processing code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requested Processing instance</td>
<td>Profile</td>
</tr>
<tr>
<td>Mobile Agent (MA Interacting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before</td>
<td></td>
<td>Required Data</td>
<td>Requested Processing instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+ Execution Status + Private Data)</td>
<td></td>
</tr>
<tr>
<td>After</td>
<td></td>
<td>Required Data</td>
<td>Requested Processing instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+ Execution Status + Private Data)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If clones:</td>
<td>Requested Processing instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+ Execution Status + Private Data)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 summarizes the components that reside at the Requesting Site and the Resource Site before and after the transfer of a process instance in each of the four mobile code desi
paradigms. Note that the table reflects the situation before Requested Processing took place, Requested Result does not yet exist. After this transfer, the executable code may be activating creating Requested Result.

4.1 Remote Evaluation

The OPD in Figure 3 is an OPM/Web model of the Remote Evaluation (REV) paradigm. Code Sending transfers an instance of Requested Processing from the Requesting Site to the Resource Site, while Code Activating invokes (triggers the execution of) this instance in the Resource Site. Finally, Requested Processing transfers the Requested Result from the Resource Site to the Requesting Site.

![A generic OPD of the REV paradigm](image)

The following OPL paragraph describes the same REV model textually.
4.2 Code-on-Demand

The OPD in Figure 4 is a generic model of the Code-on-Demand (COD) paradigm. It clearly shows that processing (i.e., the activation of a Requested Processing instance) in the COD model occurs at the Requesting Site, whereas in the REV model, shown in Figure 3, the processing takes place in the Resource Site. The fact that Requested Processing is not initially at the Requesting Site is denoted in Figure 4 by the result link (the white arrowhead) whose destination is the Requested Processing instance at the Requesting Site, indicating that the Requested Processing instance was created there only after the first stage of COD Interacting, Code Retrieving, occurred.

As described in Appendix B, OPCAT enables simulation of the behavior of this system, showing more vividly the sequence of occurrences. When the animated simulation is run, the Requested Processing instance appears only in the postcondition set of Code Retrieving.
The OPL paragraph below is the textual counterpart of the OPD in Figure 4 of the COD paradigm.

**Requesting Site** is physical.

*Requesting Site* zooms into *Activation Request*, *Required Data*, and *Requested Result*, as well as *Requested Processing* instance.

- *Activation Request* triggers **COD Interacting**.
- *Requested Processing* instance affects *Required Data*.
- *Requested Processing* instance yields *Requested Result*.

**Resource Site** is physical.

*Resource Site* zooms into *Requested Processing*.

Many *Requesting Sites* and many *Resource Sites* communicate, and this relation exhibits **COD Interacting**.

**COD Interacting** consumes *Activation Request*.

**COD Interacting** zooms into **Code Retrieving** and **Code Activating**.

- Following path *code transferring*, **Code Retrieving** requires *Requested Processing* of *Resource Site*.
- Following path *code transferring*, **Code Retrieving** yields *Requested Processing* instance of *Requesting Site*.

**Code Activating** invokes *Requested Processing* instance of *Requesting Site*.

### 4.3 PUSH

Figure 5 is a generic model of the PUSH paradigm. The following OPL sentences describe the model.
Requesting Site is physical.
Requesting Site zooms into Required Data and Requested Result, as well as Requested Processing instance.
    Requested Processing instance affects Required Data.
    Requested Processing instance yields Requested Result.
Resource Site is physical.
Resource Site zooms into Activation Request and Profile, as well as Requested Processing.
    Many Activation Requests relates to many Profiles.
    Activation Request triggers PUSH Interacting.
Many Requesting Sites and many Resource Sites communicate, and this relation exhibits PUSH Interacting.
PUSH Interacting occurs if Profile of Resource Site is requesting site.
PUSH Interacting consumes Activation Request.
PUSH Interacting zooms into Code Retrieving and Code Activating.
    Following path code transferring, Code Retrieving requires Requested Processing of Resource Site.
    Following path code transferring, Code Retrieving yields Requested Processing instance of Requesting Site.
Code Activating invokes Requested Processing instance of Requesting Site.

Figure 5. A generic OPD of the PUSH paradigm

The condition link from requesting site Profile to PUSH Interacting specifies that when triggered (by Activation Request), Requested Processing is transferred only to sites that were registered the Profile. The Activation Request and the Profile are not transferred to the Requesting Site, I only enable the transfer of Requested Processing from the Resource Site to the relev:
**Requesting Sites.** As noted, the creation of the *Activation Request* and the *Profile* at the *Resource Site* is done in a separate process whose execution precedes the execution of *PUSH Interacting*.

### 4.4 Mobile Agents

Various definitions of an agent (Franklin and Graesser, 1996) agree that all software agents are computer programs, but not all programs are agents. Each agent definition indicates some properties that differentiate an agent from a “conventional” program. Various definitions expect an agent to be reactive, autonomous, goal-oriented, temporally continuous, communicative, learning, mobile, and flexible. Agents of the same class or of different classes can communicate with each other using objects. These definitions of an agent as a computer program with additional characteristics call for modeling an OPM/Web agent as a process instance, which belongs to a process class. These process instances (agents) initiate their own migration at specific points of their execution.

![Figure 6. A generic OPD of the MA paradigm](image)

21
Figure 6 and the corresponding OPL paragraph describe a mobile agent model for the case which the agent is cloned from the Resource Site to the Requesting Site. The agent, which characterized by Private Data and an Execution Status, initiates (triggers) its own transfer with its Executing Status enters the transfer state. After completing the agent transfer, its Execution Status returns to the local state.

The instrument link from the agent (the Requested Processing instance at the Resource Site) Agent Migrating (within MA Interacting) in Figure 6 denotes that this migration clones (i.e., makes a copy of) the Resource Site’s agent at the Requesting Site. Alternatively, MA Interacting might move the agent, in which case a consumption link from Requested Processing of Resource Site to Agent Migrating replaces the instrument link, implying that the agent at the Resource Site disappears.
5. Reusing OPM/Web Code Mobility Models: The QoS System Example

In this section we demonstrate the expressive power of OPM/Web as a means to explicitly model what pieces of code are migrated along with their sources and destinations, and the effects of migration on the effectiveness of the application. Transferring a (resource or computational) component between sites involves determining the source and target sites, integrating the transferred component within the target sites, addressing network security issues, and handling errors that may occur in the process. These aspects can be incorporated in the single, bimodal graphic-textual OPM/Web model, in which one or more of the code migration models, present in the previous section, are reused. To demonstrate our approach, we present an OPM/Web model of a Quality of Service (QoS) system, a mobile application that is based on (Klein et. al., 2001). This system has been chosen in order to be able to demonstrate most of the code mobility concepts and design paradigms explained in this paper and their integration into a complete application. In this QoS system, software components from multiple parties collaborate to provide a particular service to end users. The service users access service provider hosts via a Web interface. They select the specific value-added services for their applications. A service provider communicates with several routers to achieve the QoS goals. The service users can control their requests remotely at any time.

As the System Diagram (SD), i.e., the top-level diagram, in Figure 7 shows, our QoS system consists of three types of sites: Client, ISP (Internet Service Provider) Agency, and Router Agency, each of which may have multiple instances. Each site type is modeled as a physical object that inherits from Site, a network node.

At this level of abstraction, the Client is shown to include only the QoS Interface Handling process, with which the Service User interacts. The Service User is an actor using the system at
is therefore modeled as an external (dashed) and physical (shadowed) object. Not knowing which routers provide the requested service, the Service User interacts via the QoS Interface Handling process, which the Client site hosts. This interaction is indicated in Figure 7 by the agent link (which ends with a black circle) from Service User to QoS Interface Handling. Each Client connected to ISP Agencies, and each ISP Agency is connected to several sites of type Router Agency.

![Figure 7. The top level System Diagram of the QoS System](image)

If we were to model this system with UML, we would need three different types of UML diagrams: deployment diagrams to describe the system physical architecture, use case diagrams to describe the user-system interactions, and sequence diagrams to describe scenarios of communication processes. However, even these three diagram types combined do not describe the details of the interaction processes, as do the next two OPDs in Figure 8 and Figure 9.

Refining the interaction between Client and the ISP Agency, Figure 8 shows that their communication structural relation exhibits two operations: CS Interacting and COD Interacting. The details of the models of the Client-Server (CS) and Code-on-Demand (COD) paradigms have been presented earlier. COD Interacting, for example, is the same as the process modeled in Figure 4, where ISP Agency is the server (Resource Site), Parameter Check Request is
Activation Request, and Parameter Checking is Requested Processing. Therefore, CS Interacting and COD Interacting are not in-zoomed further here.

Figure 8. Detailing the Client – ISP Agency interaction

When weaving these models into a complete application, the combined model can enhanced to handle security issues and possible transfer errors. Since the security and privacy algorithms are often pre-defined computational components, they can be modeled as OPM/Web processes, from which the transfer processes can inherit both the functionality and the interface. This open reuse mode of OPM/Web, which is beyond the scope of this paper, is described in Reinhartz-Berger, Dori and Katz (2002). The different kinds of transfer errors, such as communication failures, unknown addresses, and timeout exceptions, can be traced using OPM event links. These links model a variety of events, including process timeout, process termination, state change, state entrance, state timeout, and external events. These types of events trigger stand-alone processes, which handle the exceptions or errors as explained by Peleg and Dori (1999).

In addition to showing the details of the interaction between the Client and the ISP Agency components, Figure 8 also zooms into the Client and the ISP Agency components, exposing...
more refined view of their internal objects and processes. **QoS Interface Handling**, which is a computational component of the **Client**, handles requests that the **Service User** submits. When activated by the **Service User**, **QoS Interface Handling** creates the objects **QoS Parameter Set** and **Parameter Check Request**. Upon its creation, **Parameter Check Request** activates **COD Interacting**. The occurrence of **COD Interacting** transfers an instance (one-time executable version) **Parameter Checking** from the **ISP Agency** to the **Client**, enabling its local execution at the **Client** site. This **Parameter Checking** execution changes the state of **QoS Parameter Set** from **created** either **checked** or **wrong**, indicating whether the **QoS Parameter Set** supplied by the **Service User** is correct or wrong. Through the **QoS Interface Handling** process, the **Service User** can continue affecting **QoS Parameter Set**, in order to request services (via the **update** path) or to cancel them (via the **cancel** path). These requests are transferred to the **ISP Agency** by the **CS Interacting** process, which does not need to wait for a response from the **ISP Agency**.

Unlike UML and its extension mechanisms, OPM/Web specifies the communication processes generically, regardless of their implementation technology. For example, the **COD Interacting** process specifies a common design paradigm for code mobility without limiting it to specific implementation language constructs (such as Java applets). As this example shows, OPM/Web also supports modeling the events which trigger the communication processes, as well as the conditions that enable their activations.

Figure 9 shows a refinement of the interaction between the **ISP Agency** and the **Router Agency**. Since not all the **Router Agencies** provide all the services, the **QoS Choice Handling** uses the **Service Provider Catalog** as an instrument for creating a **Service Control Message** and the **Service Address** object, which defines a **router agency** address for the required service. If the **Service Control Message** requests a new service (which is the case when its state is **create**), then the **Router Agency**
**Interacting** process is activated, transferring an executable version of **QoS Agent Processing** the **Router Agency** according to the **Service Address**. If the **Service Control Message** is created its **update** or **cancel** states, it is transferred as is to the **Router Agency** by the **CS Interacting** process, enabling the continuous running of **QoS Agent Processing** in the **Router Agency**, where it can use the **Service Control Message** and any required **Local Data**.

![Diagram](image)

**Figure 9.** Detailing the **ISP Agency – Router Agency** interaction

Other OPM/Web code mobility models could be plugged and linked into our QoS application for specific purposes. For example, if we want **QoS Agent Processing** to be able to move or clone itself among various **Router Agencies** according to the Mobile Agent (MA) paradigm, explained in Section 4.4, we can add a structural relation between **Router Agency** and itself and specify that this relation exhibits the **MA Interacting** process as its operation.

### 6. Summary and Future Work

Existing Web and distributed system development methods are not up to the task of complete and accurate modeling of code mobility and migration. While some of them can specify static bindings of software components to their physical locations, the specification of dynamic system reconfiguration and code migration is not satisfactorily supported by any of the existing...
approaches. Mentally integrating the structure and behavior aspects of these systems in order comprehend them in their entirety can be achieved with current methods only with great difficulties due to the multiplicity of models that need to be consulted.

Using a small set of concepts and symbols, OPM/Web combines the physical, static, behavioral, and functional views of a system within a single framework. OPM/Web augments OPM to enable modeling code mobility concepts and design paradigms by specifying processes as residents of some node (site) and moving or cloning them to other nodes, where they can activated or transferred further. This approach provides for a technology-independent model where triggers, preconditions, and postconditions for the migration process are specified generically. Once the mobile application is modeled, a solid skeleton of the technology dependent implementation can be automatically generated and simulated by the Object-Process CASE Tool (OPCAT). This skeleton includes not only the structure of the application, but also its behavior, enabling design verification and leaving to the implementer only the coding at the bottom level.

The single OPM view with its combined graphic-textual modalities and abstraction-refinement mechanisms benefits from consistency, relative simplicity, and ease of learning. The structure and behavior of the different components are explicitly modeled in the same view, making them understandable and communicable. In order to model distributed applications in UML, a set of stereotypes (denoted by different graphical symbols), tagged values, and constraints must be defined. Such extension mechanisms undermine UML standardization efforts, since each researcher or company working in the domain of distributed systems is free to develop a different set of extensions. Lack of a universal set of such extension entities inhibits the efforts to develop reusable components. The segregation of a UML model into multiple views, which span acr
different diagram types, is yet another source of difficulty in capturing and understanding a system as a whole (Peleg and Dori, 2000). Indeed, comparing the complexity metric values of UML with other object-oriented techniques, Siao and Cao (2001) found that each diagram in UML is not distinctly more complex than techniques in other object-oriented methods, but as a whole, UML is 2-11 times more complex than other object-oriented methods.

In a separate work (Reinhartz-Berger and Dori, 2003), we have established the level of comprehension of a given OPM/Web model and the quality of the models constructed using it comparing OPM/Web experimentally to an extension of UML to Web applications (Conallen, 1999). Third year undergraduate information systems engineering students had to respond to comprehension and construction questions about two representative Web application models. The questions related to the system's structure, dynamics, and distribution aspects. We found that OPM/Web is significantly better in modeling the dynamics of Web applications, while specifying their structure and distribution aspects, there were no significant differences. In both case studies, the quality of the resulting OPM/Web models was superior. The main errors in UML modeling questions occurred when students were required to integrate the different views into a whole, consistent model. The modeling questions required adding a single functionality that affected several UML diagram types. All these changes were expected to leave the UML model integral and consistent. This task is difficult for trained UML modelers, let alone untrained students.

On the other hand, as our experiment indicated to some extent, UML's use of multiple views may help system architects focus on a specific aspect of a system, and answer questions about when the needed information is fully contained in a single diagram type, such as a class or interaction diagram. These types of questions may be more difficult to answer by examining
OPM/Web model, since the information might reside in several OPDs at different levels of detail. To benefit from this potential advantage of UML and to stay current with the prevailing standard, we have augmented OPCAT with the ability to automatically generate a set of UML views from the single OPM/Web model. Since UML does not have a single mechanism to express standalone processes, the resulting UML views may not necessarily be unique or completely equivalent to the OPM/Web model. Nevertheless, when we complete developing an UML OPM/Web generator, the system architect will be able to use the most suitable approach for each design portion by using OPM/Web, UML, or a combination of these two approaches. In parallel, we are working on developing the ability to generate the application (code and database schema) from the system’s OPL script.

References


http://www.cs.berkeley.edu/~franklin/Papers/datainface.pdf


http://www4.in.tum.de/~rausch/publications/2001/MobileUML.pdf


### Appendix A: Main OPM/Web Concepts, their symbols, and their meaning

<table>
<thead>
<tr>
<th>Concept Name</th>
<th>Symbol</th>
<th>Concept Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informatical object</td>
<td></td>
<td>A piece of information</td>
</tr>
<tr>
<td>Physical object</td>
<td></td>
<td>An object which consists of matter and/or energy</td>
</tr>
<tr>
<td>Process class</td>
<td>P</td>
<td>A pattern of transformation that objects undergo</td>
</tr>
<tr>
<td>Process instance</td>
<td>P</td>
<td>An executable version of code</td>
</tr>
<tr>
<td>Initial/Regular/Final state</td>
<td></td>
<td>An initial/regular/final situation at which an object can exist for a period of time</td>
</tr>
<tr>
<td>Characterization</td>
<td></td>
<td>A fundamental structural relation representing that an element exhibits a thing (object/process)</td>
</tr>
<tr>
<td>Aggregation</td>
<td></td>
<td>A fundamental structural relation representing that a thing (object/process) consists of one or more things</td>
</tr>
<tr>
<td>General structural link</td>
<td></td>
<td>A bidirectional or unidirectional association between things that holds for a period of time, possibly with a tag denoting the association semantics</td>
</tr>
<tr>
<td>Enabling event link</td>
<td></td>
<td>A link denoting an event (such as data change or an external event) which triggers (tries to activate) a process. Even if activated, the process does not change the triggering entity.</td>
</tr>
<tr>
<td>Consumption event link</td>
<td></td>
<td>A link denoting an event which triggers (tries to activate) a process. If activated, the process consumes the triggering entity.</td>
</tr>
<tr>
<td>Condition link</td>
<td></td>
<td>A link denoting a condition required for a process execution, which is checked when the process is triggered. If the condition does not hold, the next process (if any) tries to execute.</td>
</tr>
<tr>
<td>Agent link</td>
<td></td>
<td>A link denoting that a human agent (actor) is required for triggering a process execution</td>
</tr>
<tr>
<td>Instrument link</td>
<td></td>
<td>A link denoting that a process uses an entity without changing it. If the entity is not available (possibly in a specific state), the process waits for its availability.</td>
</tr>
<tr>
<td>Effect link</td>
<td></td>
<td>A link denoting that a process changes an entity. The black arrowhead points towards the process that affects the entity.</td>
</tr>
<tr>
<td>Consumption link</td>
<td></td>
<td>A link denoting that a process consumes an (input) entity. The black arrowhead points towards the process that consumes the entity.</td>
</tr>
<tr>
<td>Result link</td>
<td></td>
<td>A link denoting that a process creates an (output) entity. The white arrowhead points towards the created entity.</td>
</tr>
<tr>
<td>Invocation link</td>
<td></td>
<td>A link denoting that a process triggers (invokes) another process when it ends</td>
</tr>
<tr>
<td>XOR connection</td>
<td></td>
<td>A connection between procedural links denoting that exactly one of the process incoming/outgoing links is applicable (active) in a single execution of the process</td>
</tr>
</tbody>
</table>
Appendix B: Simulating Mobile Specifications with OPCAT

Using Object-Process CASE Tool (OPCAT)\(^3\) (Dori et al., 2003), with which the OPM models this paper were generated, a system design model can also be simulated. In the CS paradigm, example, the simulation starts by making the precondition set of the **CS Interacting** process true. This is done by enabling (through highlighting) all the components (objects and processes) which are not created by processes in the given model, i.e., the objects **Activation Request Requesting Site** and **Required Data** and the process **Requested Processing**, as shown in Figure 10(a). While executing **CS Interacting**, the **Activation Request** at the **Processing Site** becomes highlighted, then the **Requested Result** at the **Processing Site**, and finally the **Requested Result** the **Requesting Site**. After the transfer process has been completed, its postcondition set becomes true, i.e., **Requesting Site’s Requested Result**, **Processing Site’s Required Data**, and **Requested Processing** are highlighted, as shown in Figure 10(b). Using this simulation capability OPCAT, design errors that were not detected in the static model can be spotted and corrected before starting the implementation.

![Figure 10. OPCAT 2 simulation snapshots before (a) and after (b) executing CS Interacting.](image)

Existing things in a snapshot appear in grey.

---

\(^3\) OPCAT 2 can be freely downloaded from [http://www.objectprocess.org/](http://www.objectprocess.org/)