
Modelling code mobility and migration: an OPM/Web approach

Iris Reinhartz-Berger*, Dov Dori
and Shmuel Katz

Technion, Israel Institute of Technology,
Technion City, Haifa 32000, Israel

E-mail: iris@mis.hevra.haifa.ac.il E-mail: dori@ie.technion.ac.il

E-mail: katz@cs.technion.ac.il

*Corresponding author

Abstract: Web applications exhibit dynamic behaviour through such features as animation, rapidly changing presentations, and interactive forms. The growing complexity of web applications requires a rigorous modelling approach 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 satisfactorily. OPM/Web is an extension of object-process methodology (OPM) for modelling distributed systems and web applications that enables intuitive modelling of code mobility concepts in a single framework. 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 modelling such systems in OPM/Web.

Keywords: code migration; code mobility design paradigms; mobile code; object-process methodology; web application modelling.

Reference to this paper should be made as follows: Reinhartz-Berger, I., Dori, D. and Katz, S. (2005) 'Modelling code mobility and migration: an OPM/Web approach', *Int. J. Web Engineering and Technology*, Vol. 2, No. 1, pp.6–28.

Biographical notes: Iris Reinhartz-Berger is Faculty at the Department of Management Information Systems, University of Haifa, Israel. She received her PhD in Information Management Engineering in 2003 from the Technion, Israel Institute of Technology. In her MSc research she formalised conversion rules from Object-Process Language, a subset of English that constitutes the textual modality of Object-Process Methodology (OPM), to Java. Her PhD dissertation focused on developing and evaluating Web applications with object-oriented approaches and OPM. Her research interests include conceptual modelling, modelling languages and techniques for analysis and design, domain analysis, and systems development processes.

Dov Dori, PhD, is Associate Professor and Head of the Information Systems Engineering Area at the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, and a Research Affiliate at MIT, Cambridge. His research interests include Conceptual Modelling Systems Development Methodologies, Information Systems Engineering, Computer-Aided Software Engineering and Document Analysis and Recognition. Prof. Dori has developed Object Process Methodology (OPM), a holistic systems paradigm for conceptual modelling, presented in his 2002 book (by Springer). Prof. Dori is

Fellow of International Association for Pattern Recognition and Senior Member of IEEE, and he has authored over 100 journal publications and book chapters.

Shmuel Katz received his PhD in Computer Science from the Weizmann Institute of Science in 1976. He is an Associate Professor in the Computer Science Department at the Technion, Israel Institute of Technology, and heads the Systems and Software Development Laboratory there. His research interests include formal specification, verification methods, and software engineering, with over 70 papers in these areas. In recent years his work has centred on translations among verification and specification tools, and on formal methods for aspect-oriented software development. He heads the virtual Formal Methods Laboratory of the EU Network of Excellence on Aspect-Oriented Software Development.

1 Introduction

Although web applications seem to exhibit a relatively simple distributed architecture, their underlying architecture is dynamic and complex. The complexity arises from the requirements of 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 behaviour. The growing complexity of web applications requires a rigorous modelling approach. Such an 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 extends 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. Modelling the code migration process also includes defining process triggers, preconditions and post conditions, and handling security issues and possible transfer errors.

Current techniques for modelling 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 modelling 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 complicate 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 modelling the structure and behaviour 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 modelling 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 modelled in OPM/Web. Section 5 explains and demonstrates how to use these models in a complete mobile application, which handles requests for Quality of Service. Section 6 summarises and discusses OPM/Web advantages and shortcomings in modelling code migration.

2 Modelling 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 programmes 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 characterised 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 some 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 criticised as being too low-level, requiring developers to determine network addresses and synchronisation points. CS interaction is also too specific, since the client 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 CS 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 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 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 at 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 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, 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.

Discussing these design paradigms for code mobility, Carzaniga et al. (1997) claim that none of them is entirely 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 Modelling code mobility and migration

Code mobility is supported by such programming environments as Java, Telescript (White, 1996) and D'Agents (Gray et al., 2001). However, current modelling techniques

that are used in the analysis and design phases of web applications do not address code mobility concepts at a satisfactory level.

Web applications can be classified as hybrids between hypermedia and information systems (Fraternali, 1999). Most commonly, such systems are modelled using hypermedia authoring techniques or visual software engineering methods, especially object-oriented ones. Hypermedia authoring techniques, including Hypertext Design Model (HDM) (Garzotto et al., 1993), 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 modelling of the application functionality through class services and message passing among objects. Concepts involving code mobility, such as Java applets, are modelled 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 the server site), post conditions (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., 2001), 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 standardisation 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.

Behaviour-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 the modelling of dynamic configurations and the actual migration process.

OPM (Dori, 2002) combines ideas from the object-oriented development methods and behaviour-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 behaviour, the two major aspects that each system exhibits, co-exist in the same OPM model without highlighting one at the cost of suppressing the other. By integrating structure and behaviour, OPM provides a solid basis for modelling complex systems, in 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 by OPM/Web, as discussed in the next section.

3 OPM/Web and mobile components

OPM/Web extends OPM to distributed systems and especially to web applications, enabling the modelling 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 programme, 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 programme or an algorithm) is a *process instance*. The relations between things (objects and processes) are modelled by structural links (e.g., generalisation 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 interrelated Object-Process Diagrams (OPDs) constitute the graphical, visual OPM formalism. Each OPM element is denoted in an OPD by a symbol, and the 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 analysing 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 modelling code mobility concepts and design paradigms using OPM/Web.

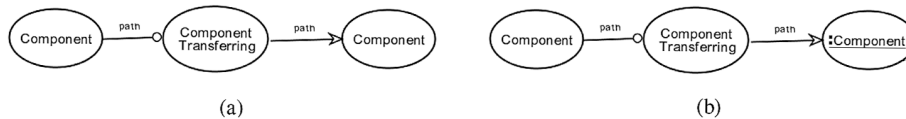
3.1 Mapping mobility terms onto OPM/Web concepts

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 sub-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 modelled in OPM/Web by a (unidirectional or bidirectional) 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 characterises the communication between the sites, the interaction process is associated in the model to the structural link that connects the two interacting sites. The implementation of this interaction may still be carried out as two interrelated 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 (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*’ transfers a process instance, i.e., only an executable version of ‘*Component*’. Following the UML notation of 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.

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.



The semantics of the arrow with the white (blank) arrowhead from ‘*Component Transferring*’ to 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 in Figure 1(b). The identical path labels² 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.

3.2 Modelling 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 by 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 or phrase and its OPD construct counterpart. The OPL paragraphs also improve system documentation.

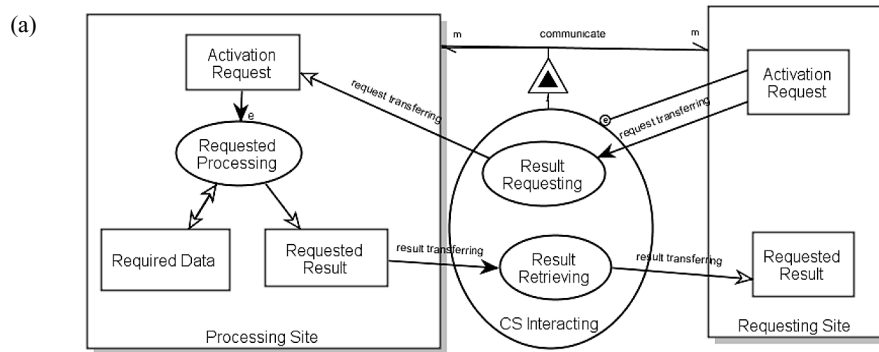
Examining Figure 2, one can see that ‘*Requesting Site*’ (the client) and ‘*Processing Site*’ (the server) are both physical objects (as denoted by shadowed rectangles). The computational component, ‘*Requested Processing*’, resides in the ‘*Processing Site*’, which also hosts the resource components 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 characterised 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 ‘*request 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.

The creation of ‘*Requested Result*’ enables the second stage of the interaction, executed by ‘*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 summarises the structure of ‘*Requesting Site*’ and ‘*Processing Site*’ before and after an activation of a ‘*CS Interacting*’ process. The dynamic aspect of the ‘*CS Interacting*’ process can be vividly simulated using OPM Case Tool (OPCAT), as explained in Appendix B.

14 *I. Reinhartz-Berger, D. Dori and S. Katz*

Figure 2 An OPM/Web model of the client–server (CS) paradigm: (a) The OPD (b) The corresponding OPL paragraph



- (b)
- Requesting Site** is 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**.

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*

<i>Design paradigm (process name)</i>	<i>Time</i>	<i>Requesting site</i>	<i>Processing site</i>
Client server (‘ <i>CS Interacting</i> ’)	Before	‘ <i>Activation Request</i> ’	‘ <i>Requested Processing</i> ’ (code) ‘ <i>Required Data</i> ’
	After	‘ <i>Requested Result</i> ’	‘ <i>Requested Processing</i> ’ (code) ‘ <i>Required Data</i> ’

4 OPM/Web models of code mobility design paradigms

OPM/Web enables precise modelling 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 for these design paradigms. In all of these models, ‘*Requesting Site*’ is the transaction client, and as 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 the process instance with a process class supports transfer of source code that can later be instantiated, i.e., compiled and executed. The various code mobility models can become generic components in specifications of mobile applications, as explained and demonstrated in Section 5.

Table 2 summarises 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 design paradigms. Note that the table reflects the situation before ‘*Requested Processing*’ took place, so ‘*Requested Result*’ does not yet exist. After this transfer, the executable code may be activated, creating ‘*Requested Result*’.

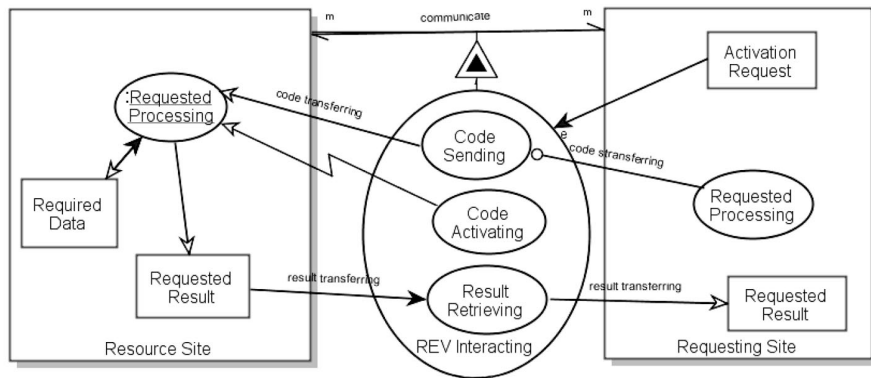
Table 2 The resource and computational components in ‘*Requesting Site*’ (the ‘client’) and ‘*Resource Site*’ (the ‘server’) before and after an activation of the transfer processes in each one of the four code mobility design paradigms

Code mobility design paradigm (process name)	Time	Requesting site	Resource site
Remote Evaluation (‘REV Interacting’)	Before	‘Activation Request’ ‘Requested Processing’ code	‘Required Data’
	After	‘Requested Processing’ code	‘Required Data’ ‘Requested Processing’ instance
Code-on-Demand (‘COD Interacting’)	Before	‘Activation Request’ ‘Required Data’	‘Requested Processing’ code
	After	‘Required Data’ ‘Requested Processing’ instance	‘Requested Processing’ code
PUSH (‘PUSH Interacting’)	Before	‘Required Data’	‘Requested Processing’ code ‘Profile Activation Request’
	After	‘Required Data’ ‘Requested Processing’ instance	‘Requested Processing’ code ‘Profile’
Mobile Agent (‘MA Interacting’)	Before	‘Required Data’	‘Requested Processing’ instance ‘Profile’ ‘(+Execution Status+Private Data)’ ‘Profile’
	After	‘Required Data’ ‘Requested Processing’ instance ‘(+Execution Status+Private Data)’	If clones: ‘Requested Processing’ instance ‘(+Execution Status+Private Data)’

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’.

Figure 3 A generic OPD of the REV paradigm



The following OPL paragraph describes the same REV model textually.

Requesting Site is physical.

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

Activation Request triggers **REV Interacting**.

Resource Site is physical.

Resource 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** of **Resource Site**.

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

REV Interacting consumes **Activation Request**.

REV Interacting zooms into **Code Sending**, **Code Activating** and **Result Retrieving**.

Following path **code transferring**, **Code Sending** requires **Requested Processing** of **Requesting Site**.

Following path **code transferring**, **Code Sending** yields **Requested Processing** instance of **Resource Site**.

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

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

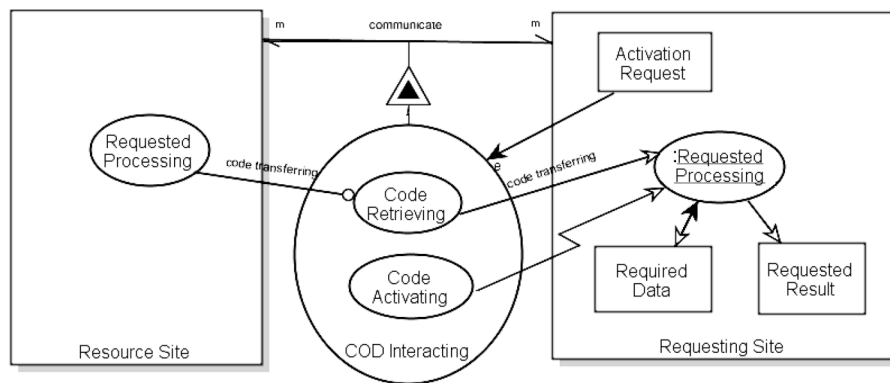
Following path **result transferring**, **Result Retrieving** yields **Requested Result** of **Requesting Site**.

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 behaviour of this system, showing more vividly the sequence of occurrences. When the animated simulation is run, the 'Requested Processing' instance appears only in the post condition set of 'Code Retrieving'.

Figure 4 A generic OPD of the COD paradigm



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**.

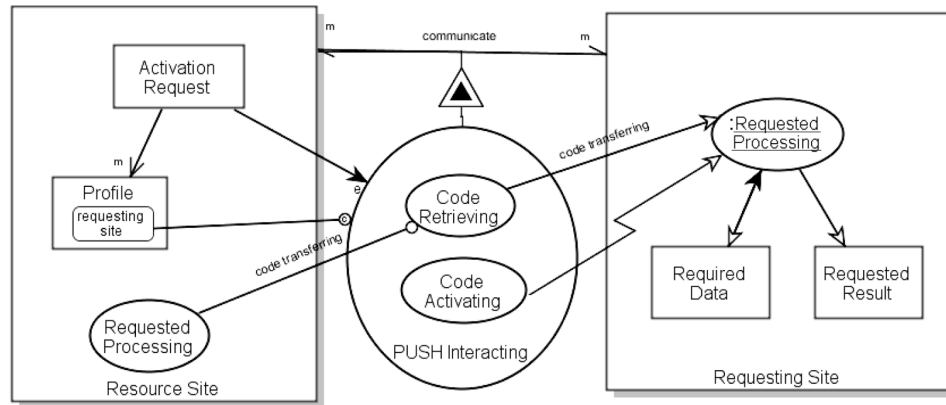
Following path **code transferring**, **Code Retrieving** requires **Requested Processing** of **Resource Site**.

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.

Figure 5 A generic OPD of the PUSH paradigm

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**.

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 in the 'Profile'. The 'Activation Request' and the 'Profile' are not transferred to the 'Requesting Site', but only enable the transfer of 'Requested Processing' from the 'Resource Site' to the relevant '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'.

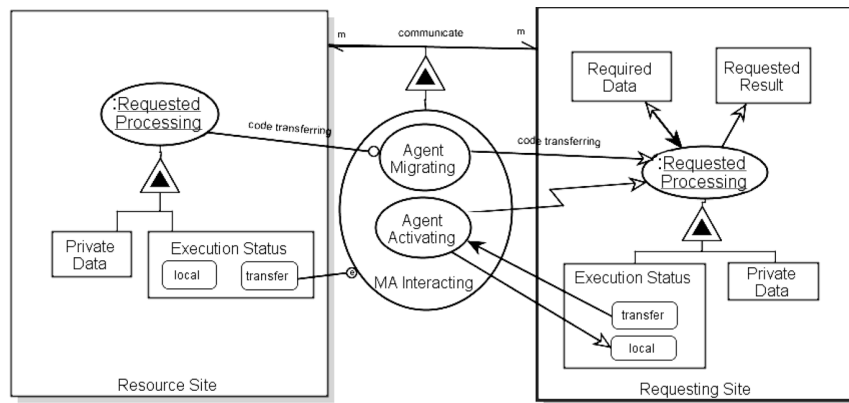
4.4 Mobile agents

Various definitions of an agent (Franklin and Graesser, 1996) agree that all software agents are computer programmes, but not all programmes are agents. Each agent definition

indicates some properties that differentiate an agent from a 'conventional' programme. 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 programme with additional characteristics call for modelling 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 and the corresponding OPL paragraph describe a mobile agent model for the case in which the agent is cloned from the 'Resource Site' to the 'Requesting Site'. The agent, which is characterised by 'Private Data' and an 'Execution Status', initiates (triggers) its own transfer when its 'Executing Status' enters the 'transfer' state. After completing the agent transfer, its 'Execution Status' returns to the 'local' state.

Figure 6 A generic OPD of the MA paradigm



Requesting Site is physical.

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

Requested Processing instance exhibits **Execution Status** and **Private Data**.

Execution Status can be **transfer** or **local**.

Requested Processing instance affects **Required Data**.

Requested Processing instance yields **Requested Result**.

Resource Site is physical.

Resource Site zooms into **Requested Processing** instance.

Requested Processing instance exhibits **Execution Status** and **Private Data**.

Execution Status can be **transfer** or **local**.

Execution Status triggers **MA Interacting** when it enters **transfer**.

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

MA Interacting zooms into **Agent Migrating** and **Agent Activating**.

Following path **code transferring**, **Agent Migrating** requires **Requested Processing** instance of **Resource Site**.

Following path **code transferring**, **Agent Migrating** yields **Requested Processing** instance of **Requesting Site**.

Agent Activating changes **Execution Status** of **Requested Processing** instance of **Requesting Site** from **transfer** to **local**.

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

The instrument link from the agent (the ‘*Requested Processing*’ instance at the ‘*Resource Site*’) to ‘*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

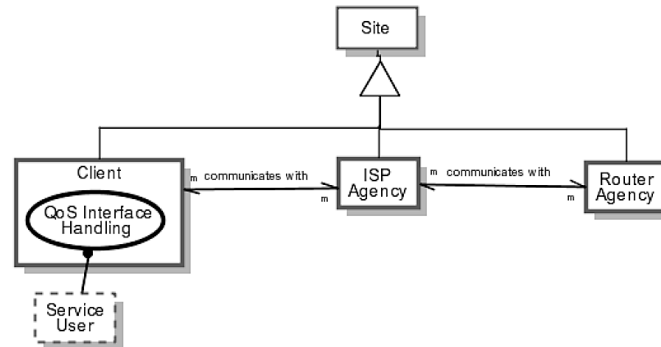
In this section we demonstrate the expressive power of OPM/Web as a means to explicitly model what pieces of code are migrated along with their sources and destinations, and the effects of the migration on the effectiveness of the application. Transferring a (resource or computational) component between 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, presented 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 Agency*’, which is the Internet Service Provider, and ‘*Router Agency*’, each of which may have multiple instances. Each site type is modelled 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 and is therefore modelled 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*’ is connected to ‘*ISP Agencies*’, and each ‘*ISP Agency*’ is connected to several sites of type ‘*Router Agency*’.

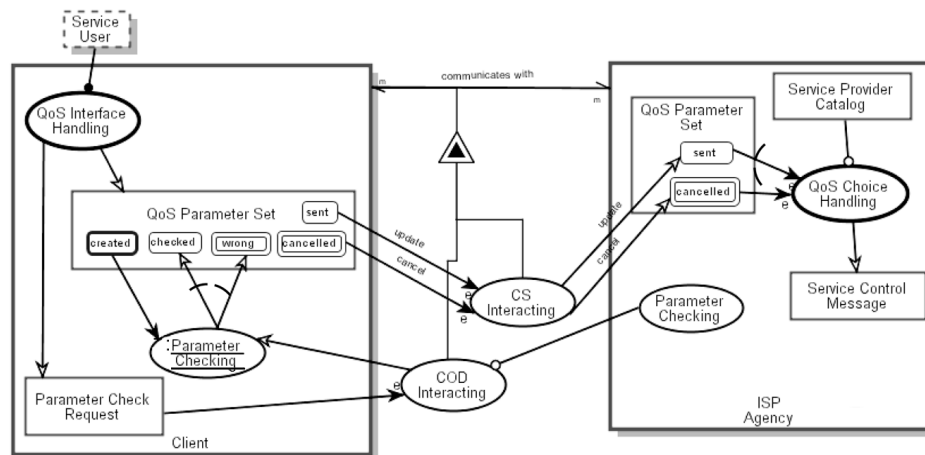
If we were to model this system with UML, we would need three different types of UML diagrams: deployment diagrams to describe the system physical architecture, use case diagrams to describe the user-system interactions, and sequence diagrams to describe scenarios of the communication processes. However,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

Figure 7 The top level System Diagram of the *QoS System*



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 modelled in Figure 4, where ‘*ISP Agency*’ is the server (‘*Resource Site*’), ‘*Parameter Check Request*’ is the ‘*Activation Request*’, and ‘*Parameter Checking*’ is ‘*Requested Processing*’. Therefore, ‘*CS Interacting*’ and ‘*COD Interacting*’ are not in-zoomed further here.

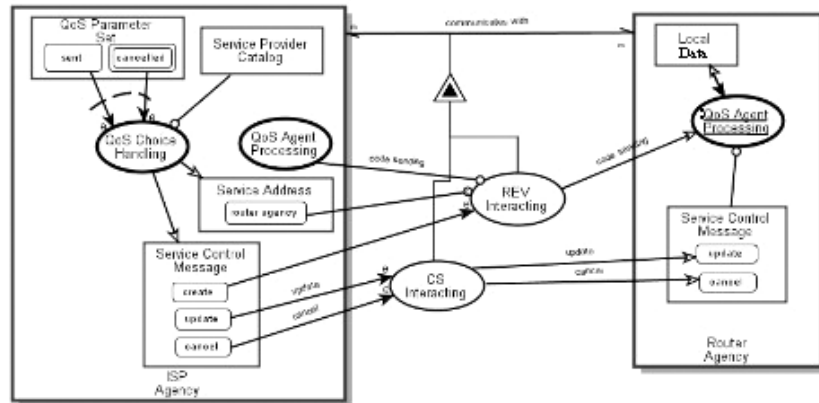
Figure 8 Detailing the *Client – ISP Agency* interaction



When weaving these models into a complete application, the combined model can be enhanced to handle security issues and possible transfer errors. Since the security and privacy algorithms are often predefined computational components, they can be modelled 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 et al. (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).

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



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 a more refined view of their internal objects and processes. '*QoS Interface Handling*', which is the 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) of '*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*' to 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 modelling 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 a '*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 'REV Interacting' process is activated, transferring an executable version of 'QoS Agent Processing' to the 'Router Agency' according to the 'Service Address'. If the 'Service Control Message' is created in 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'.

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 modelling 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 behaviour aspects of these systems in order to 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, behavioural and functional views of a system within a single framework. OPM/Web augments OPM to enable modelling 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 be activated or transferred further. This approach provides for a technology-independent model, where triggers, preconditions and post-conditions for the migration process are specified generically. Once the mobile application is modelled, 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 behaviour, 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 behaviour of the different components are explicitly modelled 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 standardisation 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 efforts to develop

reusable components. The segregation of a UML model into multiple views, which span across different diagram types, is yet another source of difficulty in capturing and understanding the 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, 2005), we have established the level of comprehension of a given OPM/Web model and the quality of the models constructed using it by 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 modelling the dynamics of web applications, while in 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 the UML modelling questions occurred when students were required to integrate the different views into a whole, consistent model. The modelling 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 modellers, 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 it when the needed information is fully contained in a single diagram type, such as a class or an interaction diagram. These types of questions may be more difficult to answer by examining an 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 stand-alone 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 to 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

- Aspect-Oriented Software Development site (2003) Available from: <http://aosd.net/>
- Bloomer, J. (1992) *Power Programming with RPC*, Sebastopol, CA: O'Reilly and Associates.
- Carzaniga, A., Picco, G.P. and Vigna, G. (1997) 'Designing distributed applications with mobile code paradigms', *Proceedings of the 1997 International Conference on Software Engineering*, pp.22–32.

- Ceri, S., Fraternali, P. and Bongio, A. (2000) 'Web modelling language (WebML): a modelling language for designing websites', *Proceedings of the 9th World Wide Web Conference (WWW9)*, *Computer Networks*, pp.137–157.
- Conallen, J. (1999) *Building Web Applications with UML*, Reading, MA: Addison-Wesley.
- Dale, J. and DeRoure, D. (1997) 'A mobile agent architecture to support distributed resource information management', *Proceedings of the International Workshop on the Virtual Multicomputer*. Available from: <http://www.mmrg.ecs.soton.ac.uk/publications/archive/dale1997b/vim97.pdf>
- Dori, D. (2002) *Object-Process Methodology – A Holistic Systems Paradigm*, Heidelberg, Berlin, New York: Springer Verlag.
- Dori, D., Reinhartz-Berger, I. and Sturm A. (2003) 'OPCAT – a bimodal case tool for object-process based system development', *5th International Conference on Enterprise Information Systems (ICEIS 2003)*, pp.286–291. Software download site: <http://www.objectprocess.org/>
- Franklin, M. and Zdonik, S. (1998) 'Data in your face: push technology in perspective', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.516–519. Available from: <http://www.cs.berkeley.edu/~franklin/Papers/datainface.pdf>
- Franklin, S. and Graesser, A. (1996) 'Is it an agent, or just a program?: A taxonomy for autonomous agents', *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, pp.21–36.
- Fraternali, P. (1999) 'Tools and approaches for developing data-intensive web applications: a survey', *ACM Computing Surveys*, Vol. 31, No. 3, pp.227–263.
- Fugetta, A., Picco, G. and Vigna, G. (1998) 'Understanding code mobility', *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp.342–361.
- Garzotto, F., Paolini, P. and Schwabe, D. (1993) 'HDM – a model based approach to hypertext application design', *ACM Transactions on Information Systems*, Vol. 11, No. 1, pp.1–26.
- Gray, R., Kotz, D., Cybenko, G. and Rus, D. (2000) 'Mobile agent: motivations and state-of-the-art systems', in J.M. Bradshaw (Ed.) *Handbook of Agent Technology*, Cambridge, MA: AAAI/MIT Press. Available from: <ftp://ftp.cs.dartmouth.edu/TR/TR2000-365.ps.Z>.
- Gray, R.S., Cybenko, G., Kotz, D., Peterson, R.A. and Rus, D. (2001) 'D'agents: applications and Performance of a mobile-agent system', *Software Practice and Experience*, Vol. 32, No. 6, pp.543–573.
- Isakowitz, T., Stohr, E.A. and Balasubramanian, P. (1995) 'RMM: a methodology for structured hypermedia design', *Communication of the ACM*, Vol. 38, No. 8, pp.34–44.
- Katz, S. (1993) 'A superimposition control construct for distributed systems', *ACM Transactions on Programming Languages and Systems*, Vol. 15, No. 2, pp.337–356.
- Klein, C., Rausch, A., Shiling, M. and Wen, Z. (2001) 'Extension of the unified modelling language for mobile agents', in K. Siau and T. Halpin (Eds) *The Unified Modelling Language: Systems Analysis, Design and Development Issues*, Hershey, PA: Idea Group Publishing Book, pp.116–128. Available from: <http://www4.in.tum.de/~rausch/publications/2001/MobileUML.pdf>
- Mezini, M. and Lieberherr, K. (1998) 'Adaptive plug-and-play components for evolutionary software development', *Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '98)*, pp.97–116.
- Muscutariu, F. and Gervais, M.P. (2001) 'On the modelling of mobile agent-based systems', *Proceeding of the 3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA'01)*, *Lecture Notes in Computer Science*, 2164, pp.219–234. Available from: <http://www-scr.lip6.fr/homepages/Marie-Pierre.Gervais/MATA2001.pdf>
- Object Management Group (1995) 'The common object request broker: architecture and specification', *Technical Report Version 2.0*. Available from: <http://www.infosys.tuwien.ac.at/Research/Corba/OMG/cover.htm>

26 I. Reinhartz-Berger, D. Dori and S. Katz



















- Object Management Group (1999) *Unified Modelling Language Specification, Version 1.3*. Available from: <http://www.rational.com/media/uml/resources/documentation/ad99-06-08-ps.zip>
- Odell, J., Parunak, H.V.D. and Bauer, B. (2000) 'Extending UML for agents', in G. Wagner, Y. Lesperance and E. Yu (Eds) *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, pp.3–17.
- Peleg, M. and Dori, D. (1999) 'Extending the object-process methodology to handle real-time systems', *Journal of Object-Oriented Programming*, Vol. 11, No. 8, pp.53–58.
- Peleg, M. and Dori, D. (2000) 'The model multiplicity problem: experimenting with real-time specification methods', *IEEE Transaction on Software Engineering*, Vol. 26, No. 8, pp.742–759.
- Reinhartz-Berger, I. and Dori, D. (2005) 'OPM vs. UML – experimenting comprehension and construction of web application models', *Empirical Software Engineering*, Vol. 10, No. 1 pp.57–80.
- Reinhartz-Berger, I., Dori, D. and Katz, S. (2002) 'Open reuse of component designs in OPM/Web', *Proceeding of Computer Software and Application Conference (COMPSAC'2002)*, pp.19–24.
- Renaud, P.E. (1993) *Introduction to Client/Server Systems: A Practical Guide for Systems Professionals*, Hoboken, NJ: Wiley & Sons.
- Schwabe, D. and Rossi, G. (1998) 'Developing hypermedia applications using OOHDM', *Electronic Proceedings of the 1st Workshop on Hypermedia Development Processes, Methods and Models (Hypertext'98)*, ACM. Available from: <http://heavenly.nj.nec.com/266278.html>
- Siau, K. and Cao, Q. (2001) 'Unified modelling language (UML) – a complexity analysis', *Journal of Database Management*, Vol. 12, No. 1, pp.26–34.
- Stamos, J. and Gifford, G. (1990) 'Remote evaluation', *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 4, pp.537–565.
- White, J.E. (1996) *Telescript Technology: Mobile Agents. Software Agents*, Cambridge, MA: AAAI Press/MIT Press.

Notes

- 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.

Appendix A

Main OPM/web concepts, their symbols, and their meaning

Concept name	Symbol	Concept meaning
Informational object		A piece of information
Physical object		An object which consists of matter and/or energy
Process class		A pattern of transformation that objects undergo
Process instance		An executable version of code
Initial/regular/final state		An initial/regular/final situation at which an object can exist for a period of time
Characterisation		A fundamental structural relation representing that an element exhibits a thing (object/process)
Aggregation		A fundamental structural relation representing that a thing (object/process) consists of one or more things
General structural link		A bidirectional or unidirectional association between things that holds for a period of time, possibly with a tag denoting the association semantics
Enabling event link		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
Consumption event link		A link denoting an event which triggers (tries to activate) a process. If activated, the process consumes the triggering entity
Condition link		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
Agent link		A link denoting that a human agent (actor) is required for triggering a process execution
Instrument link		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
Effect link		A link denoting that a process changes an entity. The black arrowhead points towards the process that affects the entity
Consumption link		A link denoting that a process consumes an (input) entity. The black arrowhead points towards the process that consumes the entity
Result link		A link denoting that a process creates an (output) entity. The white arrowhead points towards the created entity
Invocation link		A link denoting that a process triggers (invokes) another process when it ends
XOR connection		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

Appendix B

Simulating mobile specifications with OPCAT

Using Object-Process CASE Tool (OPCAT)³ (Dori et al., 2003), with which the OPM models in this paper were generated, a system design model can also be simulated. In the CS paradigm, for 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*’ (of ‘*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*’ at 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 of 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 snapshot before (a) and after (b) executing *CS Interacting*: existing things in a snapshot appear in grey

