

# Modeling Code Mobility Paradigms in OPM/Web

Iris Reinhartz-Berger, Dov Dori, and Shmuel Katz  
Technion, Israel Institute of Technology  
Technion City, Haifa 32000, Israel  
Emails: {ieiris@tx, dori@ie, katz@cs}.technion.ac.il

Each operation that involves code mobility can be divided into three steps: determining the code operation targets, transferring the code, and integrating the code into the target system. In a static system architecture, the target determination step can be done at compilation time. If the system architecture is dynamic, such that it is determined at run time, then the operation targets should be computed immediately prior to transferring the code. Following the target determination, the code is 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.

Code mobility is supported by such environments as Java, Telescript, and D'Agents. However, current analysis and design techniques do not handle this concept satisfactorily. The standard object-oriented method, UML [8], enables modeling functionality through class services and message passing among objects. Concepts involving code mobility, such as Java applets, are modeled in several views using pre-declared extensions. Hence, modeling these concepts with UML is technology-dependent (e.g., specific to the Java language and its applets) and thus can be hardly reused when other technology frames are applied. Moreover, UML does not treat the code migration process itself, including its pre- and postconditions.

In this paper we propose generic models for the transfer stage of the code migration for each one of the common design paradigms. This stage includes (1) a decision as to when and if to transfer the code and (2) its actual transfer. The models proposed in this work use OPM/Web [6], an extension of the Object-Process Methodology (OPM) [2,3] to distributed systems and Web applications. OPM/Web enables modeling the structure (i.e., objects and their relations) and behavior (i.e., processes and their links to objects) of the code migration process in a single view.

Applications that involve code mobility are defined in terms of components, sites, and interactions [1]. *Components*, which are the building blocks of system architecture, are divided into *resource components*, which are objects (architectural elements representing data, or physical devices), and *computational components*, which are processes, i.e., programs that embody flows of control. A resource component is represented by 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. *Sites* are physical objects that serve as execution environments – they host components and support the execution of computational components. *Interactions*, which are the communications between sites, have structural and dynamic aspects. The structural aspect of an interaction specifies how two sites communicate with each other irrespectively of a specific point in time, and is modeled by an OPM/Web structural link. The dynamic aspect of an interaction, which is the ability to transfer data or code between sites, is modeled by an event-triggered process that characterizes the structural link between the sites. The elementary operation of transferring a process (i.e., code) from site A to site B is shown in Figure 1. Note that the identical path labels and the identical component names indicate that the process is transferred as is (without computing it) from site A to site B.

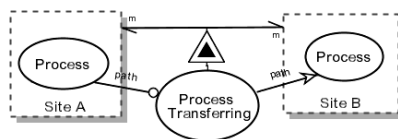


Figure 1. A generic OPM/Web model of transferring a process

Figure 2 presents the generic OPM/Web models we propose for four common design paradigms for code mobility: Remote Evaluation, Code on Demand, PUSH, and Mobile Agent. In the *Remote Evaluation (REV)* paradigm [7], a computational component is transferred from the **Client** to the **Server**, which executes the code using its resources and delivers the results back to the **Client**. Figure 2a shows that the **REV Interacting** process consists of **Code Sending**, which transfers the **Requested Processing** from the **Client** to the **Server**, **Code Activating**, which invokes **Requested Processing** in the **Server**, and **Result Retrieving**, which transfers the **Requested Result** when ready. As the event link from the **Client's Activation Request** shows, the **Client** initiates the **REV Interacting** process.

In the *Code On Demand (COD)* paradigm [1], the **Client** is able to access the resource components it needs, but has no knowledge about how to process such resource components. Thus, the **Client** interacts with the **Server**, which hosts the **Requested Processing**. Contrarily to the REV paradigm, in the COD paradigm the **Requested Processing** is executed in the **Client**. Figure 2b shows that the **COD Interacting** process consists of **Code Retrieving**, which transfers the **Requested Processing** form the **Server** to the **Client**, and **Code Activating**, which invokes **Requested Processing** in the **Client**. This invocation might change the **Required Data** and produces the **Requested Result**. Here too, the **Client** initiates the **COD Interacting** process.

In the *PUSH* paradigm [4], as opposed to the REV and COD paradigms, the **Server** sends a (computational or resource) component to the **Client** in advance of any specific request. This push-based operation is often preceded by a profiling operation, in which the **Client** specifies a

profile, which reflects its user's interests. This profile is sent to the **Server**, saved there, and used to decide what components the **Client** should receive and when to send them. The advantage of this paradigm is that the users do not have to know when to pull new components and where to pull them from. Rather, the system sends new components automatically when necessary. **PUSH Interacting**, shown in Figure 2c, is triggered due to a change in the **Requested Component** state, only if the **Profile** is set to **client** (as the event and condition links show, respectively). When activated, **PUSH Interacting** first transfers the **Requested Processing** from the **Server** to the **Client** and then activates it (in the **Client**).

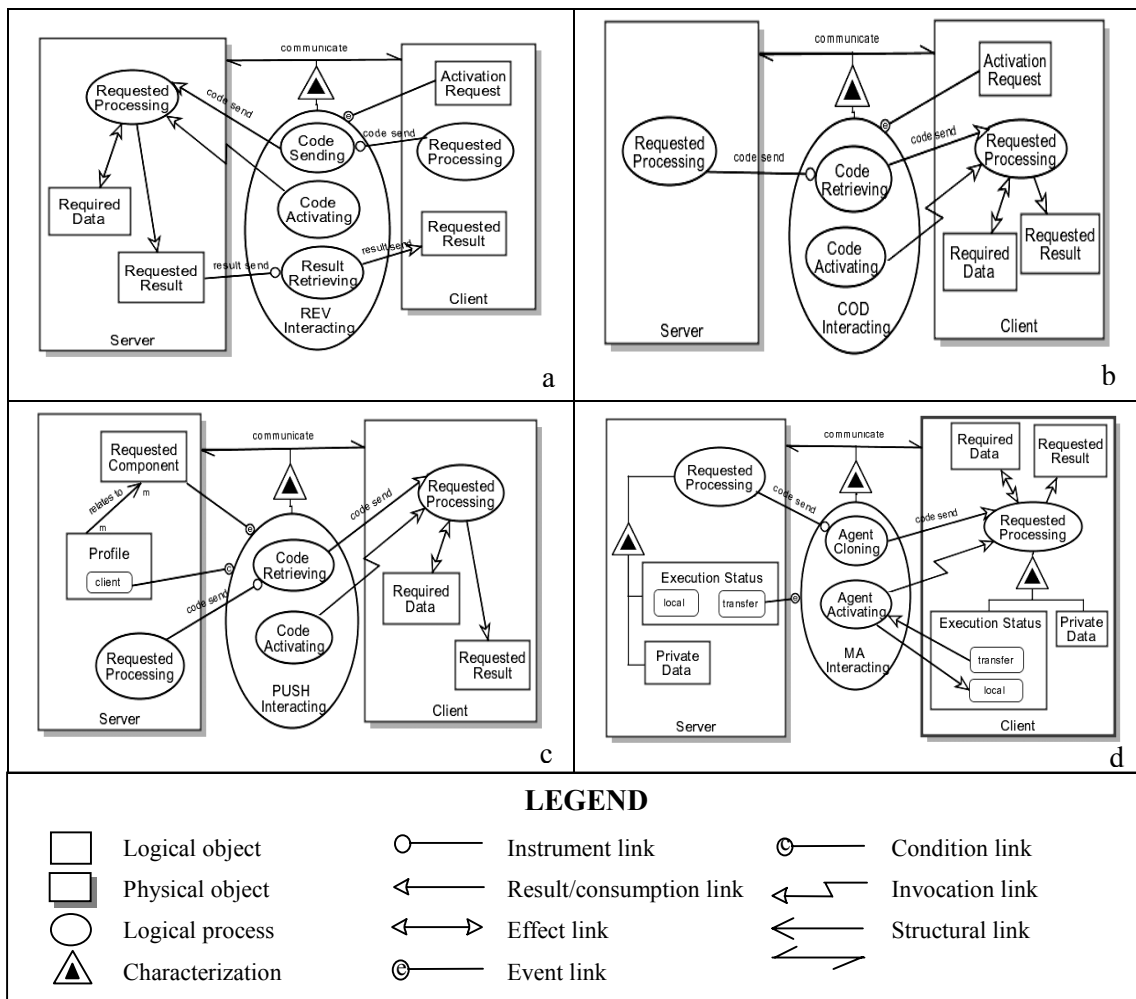


Figure 2. OPM/Web models for code mobility design paradigms. (a) The REV model. (b) The COD model. (c) The PUSH model. (d) The MA model.

In the *Mobile Agent (MA)* paradigm [5], the agent starts executing in the **Server**. Since some of the **Required Data** is located in the **Client**, the agent migrates to the **Client** and completes its execution using the resource components available there. The migration is usually initiated by the agent itself. Contrary to the previous paradigms, which focus on the transfer of just code between sites, the mobile agent migrates to the remote site as a whole computational component, along with its state, the code it needs, and some of the resource components required to perform the task. In Figure 2d the agent, **Requested Processing**,

which is characterized by **Private Data** and **Execution Status**, is hosted and executed in the **Server**. When the agent's **Execution Status** enters its **transfer** state, the **MA Interacting** process is activated (as denoted by the event link), cloning the agent to the **Client**, activating it, and changing its state to **local**. Then the agent can continue its execution in the **Client**, using additional **Required Data** and producing the **Requested Result**.

The above OPM/Web models benefit from generality and completeness and can easily be integrated into more complex systems. As opposed to UML, in which a set of stereotypes, tagged values, and constraints should be defined to support modeling these paradigms, OPM/Web uses its small set of concepts and symbols to model the paradigms in a technology-independent manner. These models can thus be implemented in Java, C#, and other programming languages without requiring changes of the core models. Moreover, OPM/Web integrates the physical, static, behavioral, and functional aspects of the code migration within a single framework, which guarantees consistency and integrity that is hard to achieve with the multiple views that UML advocates.

## References

- [1] Carzaniga, A., G.P. Picco, and G. Vigna. Designing Distributed Applications with Mobile Code Paradigms. Intel. Conf. on Software Engineering, pp. 22-32, 1997.
- [2] Dori, D. Object-Process Analysis: Maintaining the Balance between System Structure and Behavior. Journal of Logic and Computation, 5, 2, pp. 227-249, 1995.
- [3] Dori, D. Object-Process Methodology - A Holistic Systems Paradigm. Springer Verlag, Heidelberg, New York, 2002 (in press).
- [4] Franklin, M. and S. Zdonik. Data In Your Face: Push Technology in Perspective. ACM SIGMOD Intel. Conf. on Management of Data, pp. 516-519, 1998.
- [5] Gray, R., D. Kotz, G. Cybenko, and D. Rus. Mobile Agent: Motivations and State-of-the-art Systems. In Bradshaw J. M. (Ed.), Handbook of Agent Technology, AAAI/MIT Press, 2000. <ftp://ftp.cs.dartmouth.edu/TR/TR2000-365.ps.Z>.
- [6] Reinhartz-Berger, I., D. Dori, and S. Katz. OPM/Web - Object-Process Methodology for Developing Web Applications. To appear in Annals of Software Engineering: Object-Oriented Web-based Software Engineering, 2002.
- [7] Stamos, J. and G. Gifford. Remote Evaluation. ACM Transactions on Programming Languages and Systems, 12, 4, pp. 537-565, 1990.
- [8] Unified Modeling Language Specification, Version 1.3, <http://www.rational.com/media/uml/resources/documentation/ad99-06-08-ps.zip>.