IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB7407

Chapter V

Object-Process Methodology Applied to Modeling Credit Card Transactions

Dov Dori
Israel Institute of Technology, Israel

Object-Process Methodology (OPM) is a system development and specification approach that combines the major system aspects–function, structure and behavior–within a single graphic and textual model. Having applied OPM in a variety of domains, this chapter specifies an electronic commerce system in a hierarchical manner, at the top of which are the processes of managing a generic product supply chain before and after the product is manufactured. Focusing on the post-product supply chain management, we gradually refine the details of the fundamental, almost "classical" electronic commerce interaction between the retailer and the end-customer, namely payment over the Internet using the customer's credit card. The specification results in a set of Object-Process Diagrams and a corresponding equivalent set of Object-Process Language sentences. The synergy of combining structure and behavior within a single formal model, expressed both graphically and textually, yields a highly expressive system modeling and specification tool. The comprehensive, unambiguous treatment of this basic electronic commerce process is formal, yet intuitive and clear, suggesting that OPM is a prime candidate for becoming a common standard vehicle for defining, specifying, and analyzing electronic commerce and supply chain management systems.

BACKGROUND

Current object-oriented methods suffer from three major inter-related problems: the encapsulation problem, the complexity management problem, and the model multiplicity problem.

The *encapsulation problem* is a direct consequence of the OO encapsulation principle, which requires that any process be "owned" by some object, within which it is defined. While

This chapter appears in the book, Advanced Topics in Database Research by Keng Siau.

being a helpful programming convention, a direct, unavoidable consequence of this encapsulation requirement is lack of explicit process modeling. Conforming to the OO encapsulation principle suppresses the dynamic aspect of the system and imposes an unnatural modeling of the real world, because processes usually involve more than one object class. Hence, while being a suitable programming paradigm, this unnecessary encapsulation constraint has been a source of endless confusion and awkward modeling of real-life situations.

The *complexity management problem* is rooted in the fact that OO methods cope with managing the complexity that is inherent in real-life systems by breaking it into various models, one for each aspect or facet of the problem: structure (the object/class model), dynamics (Statecharts), actors (use cases), etc. When the system is large and complex, no good tools are available to seamlessly present parts of the system at varying levels of complexity.

The closely related *model multiplicity problem* stems from the fact that the fundamental OO object/class model, which is at the basis of all OO methods, is inadequate for accommodating the functional and dynamic system aspects. OO methods must employ a *host of models* to specify the various aspects of the system. The currently accepted UML standard (Fowler, 1999; OMG, 2000) requires nine different models, including class diagram, use case diagram, message trace diagram, object message diagram, state diagram, module diagram, and platform diagram.

The model multiplicity problem refers to the need to comprehend and mentally integrate a variety of models of the same system and constantly take care of synchronizing among them.

This problem arises from the requirement to concurrently construct, maintain and consult several models that represent various system aspects. Some of the confusion caused by model multiplicity is expressed in the following excerpt (Kovitz, 1998) that discusses the best mix of using UML class diagrams (the static model) and collaboration diagrams (the dynamic model):

> Class diagrams cannot stand alone. Neither can collaboration diagrams. They reinforce each other, and **need to be developed concurrently** with each other. **Failure** to develop these diagrams concurrently will **result in dynamic models that cannot be supported statically, or static models that cannot be implemented dynamically**.

We have empirically established (Peleg & Dori, 2000) that maintaining a clear and coherent image of the systems under development using such a plethora of models is a source of inherent difficulty. Comparing the major predecessor of UML–Object-Modeling Technique (Rumbaugh, Blaha, Permerlani, Eddy, & Lorsenson, 1991)), to Object-Process Methodology (OPM), we prove that an approach that is capable of specifying systems with just one model is significantly better than a multi-model one.

Object-Process Methodology

Object-Process Methodology–OPM (Dori, 1995, 2000) is a systems development approach that responds to the challenges that problems with the aforementioned OO methods raise. Using a single, integrated graphic and natural language model, OPM caters to the natural train of thought developers normally apply while trying to understand and build complex systems that involve humans, hardware and software. In such systems, it is usually the case that structure and behavior are intertwined so tightly, that any attempt to separate them is bound to further complicate the already complex description.

OPM achieves model integration by incorporating the three major system aspects–function, structure, and behavior–into a single model, in which both objects and processes