Object-Process Methodology Applied to Modeling Credit Card Transactions

DOV DORI, Massachusetts Institute of Technology and Israel Institute of Technology

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 paper 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 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 and 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 et al.), to Object-Process Methodology (OPM), we prove that an approach which 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; Dori, 2000) is a systems development approach that responds to the challenges which 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 are adequately represented without suppressing each other. This approach counters contemporary object-oriented systems development methods, notably UML, which require several models to completely specify a system. OPM is therefore not yet another OO analysis and design method, as it recognizes the fact that separating structure from behavior while engaging in system modeling, which results in the model multiplicity problem discussed above, is counter-intuitive and therefore counterproductive.

To avoid model multiplicity, OPM incorporates the static-structural and behavioral-procedural aspects of a system into a single, unifying graphic-textual model. Founded on a concise and compact ontology, in which processes and state-exhibiting objects are the only building blocks, OPM generically models the structure and behavior of physical and informational things. In the OPM ontology, objects are viewed as persistent, state preserving things (entities) that interact with each other through processes - another type of things. Thing is a generalization of an object and a process. Processes are patterns of behavior that transform objects by transforming them. Transformation is a generalization of effect, consumption and generation. Hence, transforming objects implies affecting them (i.e., changing their states), or generating new objects, or consuming existing objects. The synergy of structure-behavior unification within a single model, combined with a dual graphic-textual model, results in a highly expressive modeling tool. The OPM approach combines the graphical and textual modalities.

Graphics-Text Synergy in OPM Systems Development

OPM uses Object-Process Diagrams (OPDs) for the graphic specification and Object-Process Language (OPL) for the textual specification. This combination of graphic and text may seem redundant from a pure information-content viewpoint. In fact, however, these two modalities complement each other from the user’s perspective, because they go hand in hand such that if the diagram reader encounters some unclear point on the graphics side, she or he can directly consult the analogous textual OPL specification. Conversely, if the text is not well understood at some point along the OPL script, the corresponding OPD sentence (a construct made of one or more OPD graphic symbols) can be examined to obtain clarification. This graphics-natural language combination is a major advantage of OPM for the target audience - the professionals for whom the system is being developed. However, the same graphics-text synergy is instrumental not only for the system specification readers but also for the developers (system analysts and designers) already at the analysis and design phases.

The optimal scenario for quality systems development in terms of the professionals involved is a team comprised of one or more system architect and one or more domain experts. The domain expert knows his/her field best, but is usually not a software professional and is not supposed to be one. The system architect is proficient with systems theory and applications, but in general lacks deep knowledge of the domain within which the system is to be developed. Together, they gradually acquire knowledge about the current state of affairs surrounding the system under development. They record the knowledge accumulated using the combination of Object-Process Diagrams and Object-Process Language. When recording OPD symbols, immediate feedback is provided through OPL sentences. This enables real-time verification of the correctness of the intent and design. If the formal English sentence does not reflect the designers’ intent, immediate rectification follows. The next section describes each of the graphic and text modalities and how they relate to each other.

Object-Process Diagrams (OPDs)

Object-Process Language (OPL)

OPM uses Object Process Diagrams (OPDs), drawn using OPCAT (Dori and Sturm, 1998), for expressing the objects of a modeled system and the processes that affect them. OPCAT responds to some of the challengesJarzabek and Huang (1998) propose for current CASE tools. The OPDs