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 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 counter-productive.

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 informatical 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 natural 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) and 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 challenges Jarzabek and Huang (1998) propose for current CASE tools. The OPDs

are elaborate workflow-like hypergraphs that model the system or parts of it at various levels of detail. We present the notations and symbols OPDs use gradually as we show OPDs of supply chain management and the electronic commerce processes associated with them.

Objects and processes are connected by procedural links, which can be either enabling links or transformation links. These two different kinds of links are used in the OPD of Figure 1 to connect objects to processes, depending on the roles that these objects play in the process to which they are linked. Objects may serve as enablers - instruments or intelligent agents, which are involved in the process without changing their state. Objects may also be transformed (change their state, generated, consumed, or affected) as a result of a process acting on them.

An enabling link connects an enabler to the process that is enables. Enabler is an enabling object that needs to be present in order for the process to occur but it does not change as a result of the process occurrence. An enabling link can be an agent link or an instrument link. An agent link is denoted by a line with a black circle at the process end, such as the one from the object **Retailer** to the process **Post Product Supply & Retailing**. An agent link denotes that relative to the enabled process, the enabler is an intelligent agent - a human or an organizational unit that comprises humans, such as a department or an entire enterprise. An instrument link is an enabling link denoted by a white circle at the process end, which denotes that the enabler is an instrument - a non-human physical or informational object (machine, file, etc.) that must be present for the process to take place but is not affected by the process.

Figure 1 shows the underlying principle of how such bi-directional effect link is generated. Arrows denote transformation links, which can be effect links, consumption links and result links. Figure 1(a) shows how the process **Retailing** affects the object **Product** by transforming it from state **manufactured** to state **retailed**. In Figure 1(b), the states of **Product** are suppressed and then there is no point in keeping the incoming and outgoing effect links separate. In Figure 1(c) these two links are therefore merged into the (bi-directional) effect link.

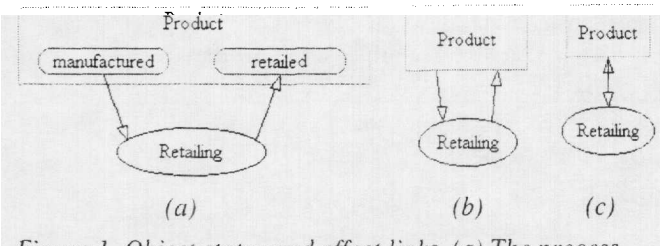


Figure 1. Object states and effect links. (a) The process Retailing affects the object Product by transforming it from state 'manufactured' to state 'retailed'. (b) States of Product suppressed. (c) The incoming and outgoing links are merged to yield the bi-directional effect link

The consumption link is a transformation link denoted as a unidirectional arrow from the consumed object to the consuming process.

Based on a constrained context-free grammar, a textual description in a natural-like Object-Process Language (OPL) can be automatically extracted from the diagrammatic description in the OPD set. Devoid of the idiosyncrasies and excess of cryptic details that feature current programming languages, OPL sentences are palatable to humans with no prior training, and can therefore be a prime candidate for becoming a business language for electronic commerce.

Applications of OPM

In this paper we present an application of OPM to modeling the basic electronic commerce process of credit card transaction as a case in point to demonstrate OPM's semantics, systems, and breads of scope. However, OPM is domain independent and, as we show below, has been applied to a large variety of domains. OPM does not depend on a particular domain of discourse but rather on fundamental definitions of an ontology that captures what an object and a process are. . Due to this generic nature of OPM, it has been found to be most suitable for developing systems in a large variety of domains. In fact, modeling of any single domain, in which OPM has been attempted as a systems development tool, has produced enlightening results. Semiconductors manufacturing and sintering technology for metal cutting tools manufacturing are two domains in which OPM has been effectively applied to produce large-scale operational systems.

Sintering technology entails mixing, pressing and "baking" rare metal powders in extremely precise conditions to obtain near diamond hardness for metal cutting. Using OPM, a team of system analysts and designers during a period of six months specified all the technological processes involved in the manufacturing of inserts. They came up with a detailed, complete specification of the system to be developed, expressed as a set of Object-Process Diagrams. The OPM design document they produced served as a blueprint for the implementation of the system. The OPD set they produced was the document that constituted the basis for contracting with the software house that implemented the MANTI system. MANTI is currently operational and serves as the backbone of the technology know-how management at a world leading insert manufacturer.

The Semiconductor Automated FAB Design Project involved 10 system analysts who used the OPM as the framework for the analysis. OPM was used by 30 in-house and 10 contractor programmers who developed the code for the system. The system was developed as an add-on to the WorkStream™ MES (Manufacturing Execution System) of the fab. The project included a detailed analysis of the fab information system. The system included as main objects the machines, the operators, the automated material handling

system and the wafers. The main processes included the major manufacturing processes (i.e. etching, photolithography, diffusion, testing, etc.), the releasing process of wafers into the fab, the transfer of wafers from machine to machine and all the transformations in the wafer status. As the project leader has noted, "The use of the OPM dramatically increased the effectiveness of the development process, since all parties (Analysts, Programmers and Users) were able to use unified terminology that covered all aspects of the system."

Other domains in which OPM has been successfully applied include studyware design (Dori and Dori, 1996), Computer Integrated Manufacturing, (Dori, 1996a) image understanding (Dori, 1996b), R&D management, (Meyersdorf and Dori, 1997), representation of control flow systems (Peleg and Dori, 1998), real-time systems (Peleg and Dori, 1999), and algorithm specification (Wenyin and Dori, 1999). In the area of document analysis and recognition, OPM was instrumental in works dealing with the Machine Drawing Understanding System and line detection (Wenyin and Dori, 1998).

The large variation of the domains listed above, in which OPM has already been successfully applied, demonstrate the generality of OPM, which makes it suitable for specifying systems independently of their domain of discourse. In particular, as we show in this paper, modeling supply chain and electronic commerce systems with OPM provides complete and concise specification that is palatable to software and domain experts alike.

Credit card transactions

Figure 2, adapted from Textcor (1998), describes credit card processing in a free natural language, while an accompanying pictorial scheme provides the details of electronic shopping that precedes the credit card processing.

The description in Figure 2 concerns details of the **Retailing** process, which is the focus of the OPD depicted in Figure 3 and its corresponding OPL paragraph in Figure 4. The two modalities - the graphic specification through the OPD and the textual one through the OPL - complement each other and reinforce the clarity of the system specification. The structure and behavior of **Retailing** are self-explanatory to the extent that there is not much to be added without simply repeating the content of the OPL sentences. In what follows, we examine the system specification obtained so far to study several unique features of OPM.

What is involved in credit card processing?

The steps in credit card processing are as follows.

Authorisation

The merchant must first obtain authorisation for the charge from the merchant's credit card processing company. Authorisation simply means that the card has not been reported stolen, and there is sufficient credit on the card. It results in the customer's credit limit being temporarily reduced by the value of the transaction.

There are two ways in which authorisation may be obtained:

1. Manual: The merchant downloads details of the sale from the computer that is acting as web server. The merchant then requests authorisation using their normal method such as a point of sale (POS) terminal or PC program.

2. Automatic: The server software communicates directly with the credit card processing company computer and arranges authorisation on-line.

Clearly option 2 is preferred, but this is more complex and the costs are greater.

Capture

The final stage is for the credit card to be debited. This can happen at the same time as authorisation provided the merchant guarantees that delivery will take place within a certain fixed time. Otherwise capture should take place when the goods are shipped.

If the merchants business is such that capture can take place immediately, then this can also happen automatically. Otherwise a second manual process is required.

Charge back

Regretfully there is sometimes a further stage where the customer is dissatisfied and arranges for the transaction to be cancelled. Because many Internet sales are made to overseas customers many banks perceive that there is an increased risk of charge backs. It has been reported that some merchants will not accept orders to Russia because of the frequency of charge back.

Note that the fact that a payment has been authorised by the bank does not provide any protection against charge-back.

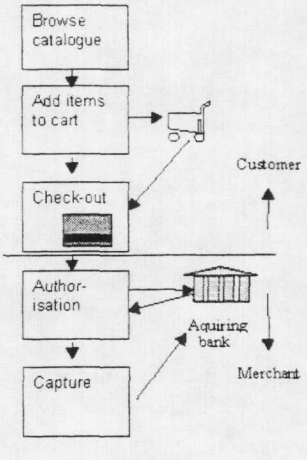


Figure 2. Natural language and schematic description of electronic shopping and credit card processing

Modeling prose specification

While modeling a system (or "problem statement") given in prose, some details are not modeled in the OPM model, while others that are not explicit in the prose statement are expressed in an explicit manner. Thus, in Figure 2, the word "*Regretfully*" in the sentence "*Regretfully there is sometimes a further stage where the customer is dissatisfied and arranges for the transaction to be cancelled.*" is not modeled, since it is a state of mind that is not very relevant in the model. This is where Actor Network Theory, discussed below, could be useful.

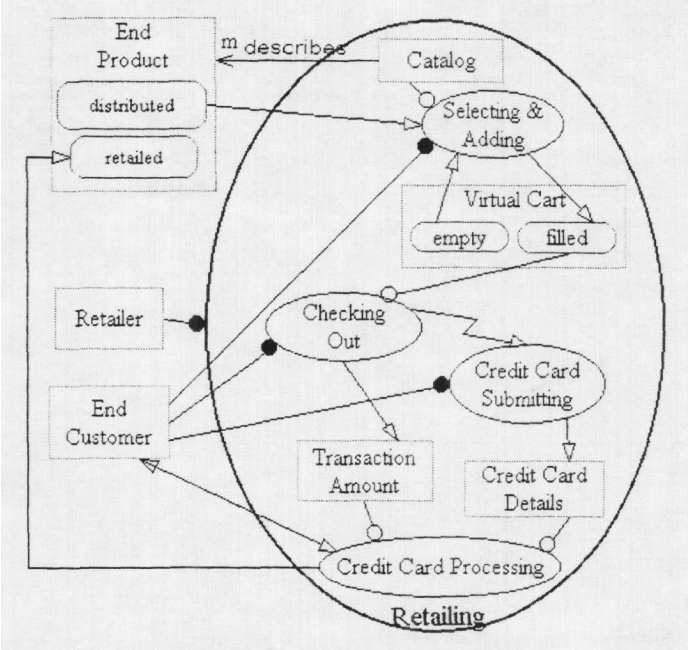


Figure 3. Zooming into the Retailing process

Retailer handles **Retailing.**
Retailing zooms into **Selecting & Adding**, **Credit Card Submitting** and **Credit Card Processing.**
Retailer handles **Selecting & Adding.**
Selecting & Adding requires **Catalog.**
Catalog describes many **End Products.**
Selecting & Adding changes **End Product** from **distributed.**
Selecting & Adding changes **Virtual Cart** from **empty** to **filled.**
Checking Out occurs if **Virtual Cart** is **filled.**
End Customer handles **Checking Out.**
Checking Out yields **Transaction Amount.**
End Customer handles **Credit Card Submitting.**
Credit Card Submitting yields **Credit Card Details.**
Credit Card Processing requires **Credit Card Details.**
Credit Card Processing changes **End Product** to **retailed.**

Figure 4. The OPL paragraph of the OPD in Figure 3

Structural links and fundamental structural links

The word **describes** in the OPL sentence

Catalog describes many **End Products.**

of Figure 4 is written in the OPD of Figure 3 along an arrow with an open arrowhead, which symbolizes a (general, or tagged) structural link. Unlike procedural links, which connect a process with an object or object state, structural links connect one object to another or one process to another. The name of the relation, **describes** in our case, is written such that a legal natural English sentence is produced when the source object (e.g., **Catalog**), the relation name (tag) and the destination object (**End Product**), are listed in this order. The word many reflects the "m" next to the arrowhead in Figure 3, which symbolizes a one-to-many participation constraint. Aggregation, generalization, characterization and instantiation are *fundamental structural relations.*

Fundamental structural relations are the four structural relations which, due to their frequent usage, are denoted by special (triangular) symbols. Characterization, for example, is denoted by a black-on-white triangle, as shown in Figure 5 and translated to the OPL sentence

End Customer features **Satisfaction.**

Credit Card Processing is represented in Figure 3 as a sub-process of **Retailing**. To model the free text description of Figure 2, in Figure 5 we zoom into **Credit Card Processing**, and in Figure 6 we provide the corresponding OPL paragraph. As before, the combination of the OPD and OPL is almost self-explanatory.

Control structures

Figure 5 demonstrated two basic control structures: if-then and if-then-else. The object **Satisfaction** and its states exemplify the if-then statement.

The OPL sentence is

Back-Charging occurs if **Satisfaction** is **low.**

The object **Authorization** and its states exemplify two if-then statements:

Capturing occurs if **Authorization** is **granted.**
Notifying occurs if **Authorization** is **denied.**

Since there are only two **Authorization** values, these two OPL sentences could be combined into the following single OPL sentence:

Capturing occurs if **Authorization** is **granted,**
else **Notifying** occurs.

A case statement is a generalization of these examples, where the number of states is not limited to two. In Peleg and Dori (1998) we show how other control structures, like loops and recursion, can also be explicitly modeled within OPDs.

Distributing procedural and structural links

A link from an enabling object to the circumference of a zoomed-in process implies that the link is attached to each

one of the sub-processes within the zoomed in process. Thus, for example, the instrument link from the object **Transaction Amount** to the abstract, higher-level process **Credit Card Processing** implies that **Transaction Amount** serves as instrument for all four lower-level processes **Authorizing**, **Capturing**, **Notifying** and **Back-Charging** that comprise **Credit Card Processing** and are depicted within its zoomed-in enclosing ellipse. The enclosing high-level process acts as a graphic shorthand notation that saves drawing many links from the enabling object to each one of the enabled processes.

This is analogous to a pair of parentheses in an algebraic expression that act to shorten the expression through the distributive law, where $a\Re(b+c+d+e)$ is equivalent to $a\Re b+ a\Re c+ a\Re d+ a\Re e$. In this case, a is the object **Transaction Amount**, $\Re$ is the instrument link, and b, c, d and e are the four processes **Authorizing**, **Capturing**, **Notifying** and **Back-Charging** that comprise **Credit Card Processing**. More generally, $\Re$ can be any relation or operation that obeys the distributive law and a, b, c, d and e the proper operands.

The distributive nature of relations exists not only for procedural relations but also for structural ones. Thus, in Figure 8, the structural relation **holds** emanating from **Web-Server** is common to both **Transaction Amount** and **Credit Card Details**. Therefore, in Figure 9, instead of writing the two separate sentences

Web-Server stores Transaction Amount.

and

Web-Server stores Credit Card Details.

we write shortly, in one sentence:

Web-Server holds Transaction Amount and Credit Card Details.

It should be noted that this is a simplified model as it does not take into account the distinction between the card issuing bank and the acquiring bank - the bank that has a business relationship with a merchant and receives all credit card transactions from that merchant. (Rosenberg, 1993).

Our model does not include the Interchange Fee - A fee the acquiring bank pays to the credit card issuing bank in order to process a credit card transaction involving a card holder's account (Rosenberg, 1993). The Merchant Discount, which is a percentage of the retail sale the merchant pays as a fee to the acquiring bank for processing the credit card transaction is not accounted for either. The more accurate model, based on (Lamond, 1996) appears in Figure 7. For the sake of brevity, we refer in this model to both the acquiring bank and the credit card issuing bank as the "**Credit Card Processing Company**".

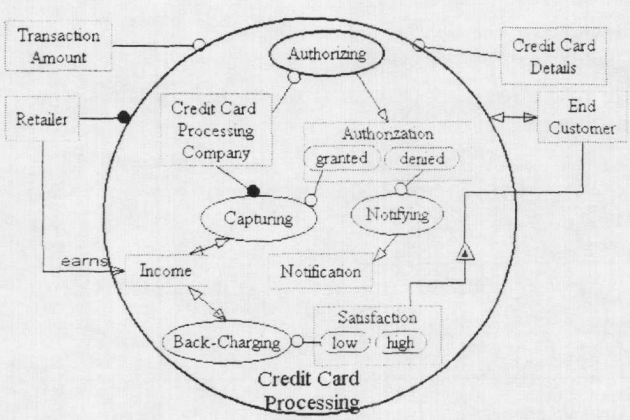


Figure 5. Zooming into Credit Card Processing of Figure 3

Paths, use cases and threads of execution

Figure 8 and Figure 9 deal with the Authorization process. In the narrative text that describes the system (Figure 2) we find that there is a manual option and an automatic one:

> There are two ways in which authorisation may be obtained:
> 1. **Manual:** The merchant downloads details of the sale from the computer that is acting as web server. The merchant then requests authorisation using their normal method such as a point of sale (POS) terminal or PC program.
> 2. **Automatic:** The server software communicates directly with the credit card processing company computer and arranges authorisation on-line.

We denote these two alternatives as two different paths, marked **Manual** and **Automatic**. A *path* in an OPD is a collection of procedural relations. Together they denote a *use case* - a possible scenario or happening, or part of a scenario

> **Retailer** handles **Credit Card Processing.**
> **Credit Card Processing Company** handles **Credit Card Processing.**
> **Credit Card Processing** requires **Transaction Amount.**
> **Credit Card Processing** requires **Credit Card Details.**
> **Credit Card Processing** affects **End Customer.**
> **Credit Card Processing** zooms into **Authorizing, Capturing, Notifying** and **Back-Charging.**
> **Authorizing** yields **Authorization.**
> **Authorization** can be **granted** or **denied.**
> **Notifying** occurs if **Authorization** is **denied.**
> **Notifying** yields **Notification.**
> **Capturing** occurs if **Authorization** is **granted.**
> **Capturing** affects **Income.**
> **Retailer earns Income.**
> **End Customer** exhibits **Satisfaction.**
> **Satisfaction** can be **high** or **low.**
> **Back-Charging** occurs if **Satisfaction** is **low.**
> **Back-Charging** affects **Income.**

Figure 6. The OPL paragraph of the OPD in Figure 5

in the systems that needs to be differentiated from one or more use cases.

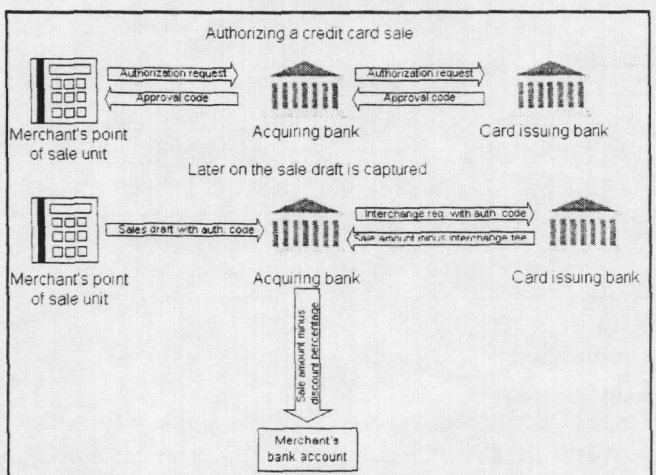
Boolean objects

A Boolean object is an object with exactly two states (values): yes and no. Its name is phrased as a statement containing the reserved word is and ending with a question mark, which uniquely identifies a Boolean object. There are two Boolean objects in Figure 10: "**Credit Card** is **Reported Stolen?**" and "**Credit Limit** is **Exceeded?**" These have been introduced to model the system specification of Figure 2 that reads:

> Authorisation simply means that the card has not been reported stolen, and there is sufficient credit on the card.

The OPL sentence pair that refers to a Boolean object are phrased so as to make sense in a natural language. For the "**Credit Card** is **Reported Stolen?**" Boolean object, the two OPL sentences, which appear in Figure 11, are as follows:

Legal Use Denying occurs if **Card** is **Reported Stolen.**
Credit Limit Checking occurs if **Card** is not **Reported Stolen.**



Merchant transmits the credit card data and sales amount with a request for authorization of the sale to their acquiring bank. The acquiring bank that processes the transaction routes the authorization request to the card-issuing bank. The credit card number identifies type of card, issuing bank, and the cardholder's account. If the cardholder has enough credit in their account to cover the sale, the issuing bank authorizes the transaction and generates an authorization code. This code is sent back to the acquiring bank. The acquiring bank processing the transaction and then sends the approval or denial code to the merchant's point of sale unit. Each point of sale device has a separate terminal ID for credit card processors to be able to route data back to that particular unit.

Figure 7. A more detailed model of the authorization and capturing processes that includes the interaction between the acquiring bank and the card issuing bank

Application generation from the OPL specification

Figure 12 specifies precisely the following system requirement denoted in Figure 2:

It (authorization) results in the customer's credit limit being temporarily reduced by the value of the transaction.

Figure 12 and Figure 13 are the pair that specifies the **Limit Checking & Decreasing** process.

Summarizing the Credit card transaction system specification, we see that we started with **Retailing** - a very broad system definition. Following a series of refinements through zooming into processes and unfolding the associated objects, we ended up with a very concrete, down-to-earth minute detailed processes such as decreasing the credit limit of an end customer. Theses can be automated to generate code.

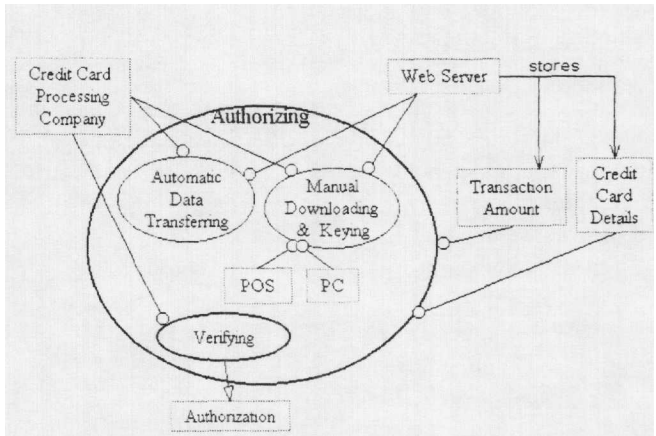


Figure 8. Zooming into Authorizing of Figure 5

> **Web-Server holds Transaction Amount** and **Credit Card Details.**
> **Authorizing** zooms into **Manual Downloading & Keying, Automatic Data Transferring** and **Verifying.**
> **Manual Downloading & Keying** requires **Web-Server** and **Credit Card Processing Company.**
> **Automatic Data Transferring** requires **Web-Server** and **Credit Card Processing Company.**
> **Credit Card Processing Company** enables either **Manual Downloading & Keying** or **Automatic Data Transferring.**
> **Web-Server** enables either **Manual Downloading & Keying** or **Automatic Data Transferring.**
> **Manual Downloading & Keying** requires either **Point POS** or **PC.**
> **Verifying** requires **Transaction Amount** and **Credit Card Details.**
> **Verifying** yields **Authorization.**

Figure 9. The OPL paragraph that corresponds to the OPD in Figure 8

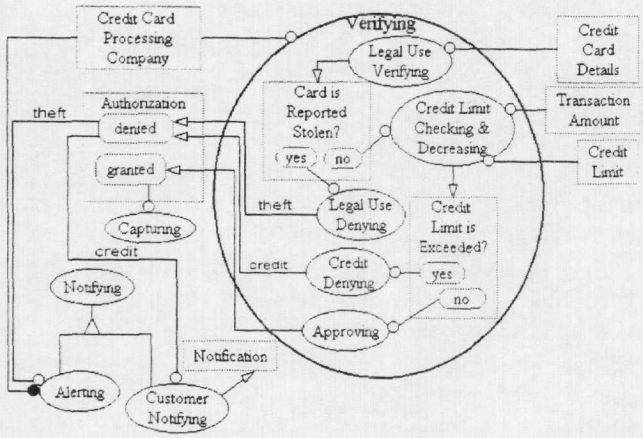


Figure 10. Zooming into Credit Card Processing of Figure 8

Verifying requires **Credit Card Company.**
Verifying zooms into **Legal Use Verifying, Legal Use Denying, Credit Limit Checking,**
Credit Denying and **Approving.**
Legal Use Verifying requires **Credit Card Details.**
Legal Use Verifying determines whether **Card** is **Reported Stolen.**
Legal Use Denying occurs if **Card** is **Reported Stolen.**
Credit Limit Checking & Decreasing occurs if **Card** is not **Reported Stolen.**
Credit Limit Checking & Decreasing requires **Transaction Amount** and **Credit Limit.**
Credit Limit Checking & Decreasing determines whether **Credit Limit** is **Exceeded.**
Credit Denying occurs if **Credit Limit** is **Exceeded.**
Approving occurs if **Credit Limit** is not **Exceeded.**
Credit Denying yields **denied Authorization** along the **credit** thread.
Legal Use Denying yields **denied Authorization** along the **theft** thread.
Approving yields **granted Authorization.**
Notifying generalizes **Alerting** and **Customer Notifying.**
Customer Notifying requires **denied Authorization** along the **credit** thread.
Customer Notifying yields **Notification.**
Alerting requires **denied Authorization** along the **theft** thread.
Credit Card Company handles **Alerting.**
Customer Notifying yields **Notification** along the **credit** thread.
Capturing requires **granted Authorization.**

Figure 11. The OPL paragraph of the OPD in Figure 10

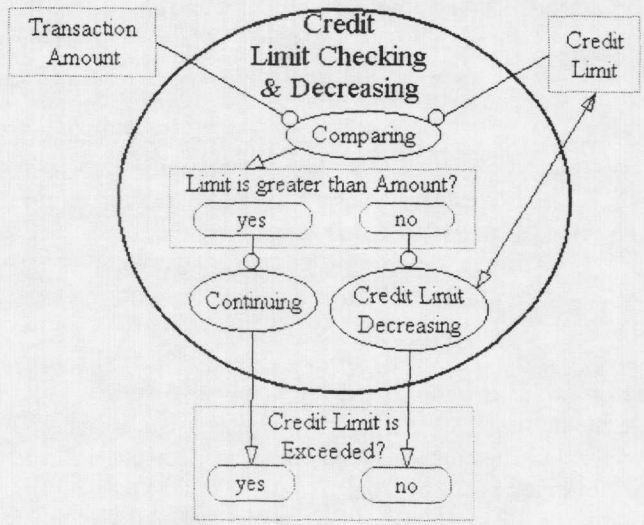


Figure 12. Zooming into Credit Limit Checking

Limit Checking & Decreasing zooms into **Comparing, Credit Limit Decreasing** and **Continuing.**
Comparing requires **Transaction Amount** and **Credit.**
Comparing determines whether **Limit** is greater than **Amount.**
Credit Limit Decreasing occurs if **Limit** is not greater than **Amount.**
Continuing occurs if **Limit** is greater than **Amount.**
Credit Limit Decreasing affects **Credit Limit.**
Credit Limit Decreasing establishes that **Credit Limit** is not **Exceeded.**

Figure 13. The OPL paragraph of the OPD in Figure 12

OPM and Actor-Network Theory

Actor-Network Theory (ANT) was born out of ongoing efforts within the field called social studies of science and technology and developed from studies in two related but distinct fields: the social practice of science and the introduction of new technologies. An early paper by Latour and Woolgar (1979) looks at struggles over scientific truth in a laboratory, while one of Callon's early studies (1986) considers fishermen and scallops as some of the stakeholders in a changing fishing industry. These examples already exhibit some of the main features of ANT.

The underlying idea of ANT is that business is never done in a total vacuum but rather under the influence of a wide range of surrounding factors. The actors may be humans, organizations, cultures, ideas, animals, plants or inanimate objects, and these are treated symmetrically irrespective of their ontology. These actors have interests which are represented (in both the semiotic and political senses) by themselves and other actors. In line with its semiotic origin, actor

network theory is granting all entities of such a heterogeneous network the same explanatory status (Akrich and Latour 1992, p.259).

ANT and OPM share in common breadth of scope that extends well beyond information systems in their traditional sense. Both ANT and OPM view the entire universe as the stage where existence and action take place. ANT actors are OPM objects, and ANT actions are OPM processes.

The act an ANT actor carries out and all of the influencing factors should be considered together. The actor-network is a shifting system of alliances and exchanges among the actors. It is the act linked together with all of its influencing factors (which again are linked), producing a network (Hanseth and Monteiro, 1998). An actor network consists of and links together both technical and non-technical elements. Not only the car's motor capacity, but also one's driving training and conditions influence the driving. Hence, ANT talks about the heterogeneous nature of actor networks.

In OPM terms, an ANT network is the system defined by an OPM specification, which is expressed by two completely equivalent modalities: a set of Object-Process Diagrams - OPD set, and an equivalent collection of Object-Process Language sentences - the OPL script. Like ANT, OPM can incorporate non-technical objects and processes, including humans, political or industrial organizations, and any kind of inanimate item or living organism along with their structure and behavior. Behavior is the dynamics of each object and of the system as a whole. It is exhibited as a transformation in one or more objects in the system, which can be a change in an object's state, its generation or its consumption.

Akrich and Latour (1992) claim several advantages for the ANT approach. It is symmetrical with respect to type of actor: it treats humans and machines equally; it is symmetrical with respect to outcome: failures have the same types of explanation as successes; and it is symmetrical with respect to causality: each actor influences and is influenced by other actors and the network as a whole.

The equal treatment of people and machines has been criticized but Underwood (1998) suggests that it may be realistic in terms of power relations and it prevents issues from becoming invisible when their representation is transferred (translated) to an actor of a different type. If, for example, some data collection functions are transferred from police informers to computer programs it is still important to be able to talk about the power relations and motives of the collectors and their allies.

Like ANT, OPM provides the system developer with facilities to account for actors, which are the OPM enablers - agents and instruments. Unlike ANT, though, agents are humans or organizations with intelligence and intent, while instruments are any physical or informatical objects that are non-human and therefore are not characterized by free will and intent. While it may be important to be able to talk about the power relations and motives of the collectors and their allies in the above example, the computer that stores that information is a mere instrument that cannot, on its own, make any political use of the information is stores. In OPM, a clear distinction is therefore made between the two enabling types. Moreover, while an OPM process transforms (affects, consumes or generates) at least one object, enablers do not change as a result of the occurrence of a process.

The interests of ANT actors are represented by scripts, usually imperative statements such as "shut the door", "pay your taxes" or "calculate the gross pay". Akrich and Latour (1992) give a comprehensive set of definitions of script-related processes (such as inscription and conscription). These processes describe (amongst other things) the translation of scripts among actors, often involving a change of medium, for example from conditioned response in a human to lines of code in a computer program.

ANT scripts are OPM processes in the sense that they specify what transformation an object undergoes. Thus, the script "shut the door" is like the Door Shutting" process in OPM, where some enabler (agent or instrument) changes the state of the object Door from "open" to "closed". Likewise, the "pay your taxes" ANT script is the "Tax Paying" OPM process, where an agent (the object Citizen) changes the state of the object Tax from "unpaid" to "paid". Gross Pay Calculating" is the OPM process of the ANT script "calculate the gross pay". Here, an instrument (the object Computer) or an agent (the object Clerk) changes "Gross Pay" from "unknown" to "calculated".

Of particular interest is ANT's idea of description (description), the discovery of the words behind the things or actions. This discovery is only possible in contrived, exotic or crisis situations, such as reengineering, consultancy or system failure; in a time such as IS development when nothing is taken for granted. Indeed OPM' users experience a similar phenomenon. While trying to explicitly determine all the processes and objects with their states in the system even the most experienced domain experts frequently find themselves in a situation where they need to define and invent names for things that are central to the domain of discourse. These are things with which the domain experts had been working for years without giving themselves or others account about the exact nature of those things.

Scripts are imperative but don't have intentions; actors do. An actor can develop a "program of action" (Akrich and Latour, 1992) perhaps with the intention of maximising the number of actors following a particular script. Some actors may avoid this by following an anti-program. A program of action can include the creation of new actors suitably inscribed. The inscription is most effective if it becomes irreversible, if the actor is, with respect to that script, a "black box" and the script becomes inaccessible to other actors.

ANT helps us to understand the course of a project or enterprise. We can asks questions such as "How did it come

to turn out this way?" (through the changing alliances of actors), "Who is influencing it?" (who has been doing what scripting?) or "Why are some actors acting this way? " (what scripts are they carrying?). OPM goes a long way in pinpointing cause and effect. Consider the OPD of Figure 10. One can tell exactly why, for example, authorization for some transaction was denied.

Some of the more spectacular applications of ANT have been to the genealogy of now well established scientific theories (Latour, 1987), the meaning of simple technical devices (Latour, 1992) or the acceptance of a new product (Bijker, 1992). More recently ANT has been applied to the development of information systems. Monteiro and Hanseth (1996) claim that ANT allows a finer grained analysis of information systems than some other interpretive approaches which can treat all information systems as essentially similar.

The majority of IS development projects still follow, albeit loosely (Fitzgerald, 1997), methodologies derived from the systems approach to problem solving popularized by the RAND corporation in the 1950s (Optner, 1973). This approach takes us through the steps of problem definition, search for solutions, selection of the best alternative, implementation and evaluation. As with many methodologies what starts as a description of how particular projects were done soon becomes a normative or prescriptive theory, which promises future success. This may be a reasonable transformation if the original projects were successful, but after 40 years of IS development several factors encourage IS researchers to look for different theoretical bases for methodologies (Underwood, 1998).

Firstly, a large percentage of computer based information systems are generally acknowledged to provide less than satisfactory service to end-users and to fall short of their original objectives. While some authors have attributed these failures to developers not following accepted theories, there is always the suspicion that the theories themselves may be at fault (Beath and Orlikowski, 1994). OPM can be a concrete framework that serves as a unifying theory in itself, as well as a platform with which more abstract (but less strictly defined) theories, such as ANT, can be explicitly expressed and discussed.

To summarize this section, a comparison between OPM and ANT has revealed a host of common features between the two approaches. OPM can be a prime vehicle to express advanced ANT ideas and serve as a basis for developing and proving ANT claims and hypotheses.

Summary

This paper has presented the Object-Process Methodology (OPM) as a viable approach for precise and explicit specification of complex systems, such as manufacturing, enterprise resource planning, electronic commerce and supply chain. To demonstrate the relevance of OPM to electronic commerce we first analyzed a broad, generic supply chain management system and its supporting electronic commerce infrastructure. Gradually, we narrowed our focus to the electronic commerce aspects of the system. In particular, we focused on the final stages of electronic commerce that takes place between the retailer and the customer. Within the retailing process we further focused on credit card based payment, which is a broadly practiced electronic commerce activity. While noting that business-to-business electronic commerce can also practice this method of payment, we restricted the analysis to the interaction between the end customer and the retailer with the involvement of the credit card processing company.

OPM appeals to the human intuition as it combines system structure and behavior into a single model that caters to the natural train of thought humans apply while trying to understand a complex system. As shown throughout the paper, the synergy of combining structure and behavior using the formal graphic representation along with a corresponding equivalent textual representation yields a system specification tool with high expressive power. Due to these virtues of formality on one hand and intuitiveness on the other hand, Object-Process Methodology is most suitable as an infrastructure for Internet-based systems engineering environment, within which inter-corporate business processes, such as ERP, and other electronic commerce activities can be seamlessly defined and conducted.

References

Akrich, M. and Latour, B. (1992). A Summary of a Convenient Vocabulary for the Semiotics of Human and Nonhuman Assemblies. in Bijker, Wiebe E. and Law, John (Eds) *Shaping Technology / Building Society*, MIT Press, Cambridge, MA.

Beath, C.M. and Orlikowski, W.J. (1994). The Contradictory Structure of Systems Development Methodologies. *Information Systems Research* 5(4), 350-377.

Booch, G., Jacobson, I., and Rumbaugh, J. (1997). *Unified Modeling Language (UML) Notation Guide Version 1.1*, Rational Software Corporation.

Dori, D. (1995). Object-Process Analysis: Maintaining the Balance between System Structure and Behavior. *Journal of Logic and Computation*, 5(2), 227-249.

Dori, D. (1996a). Object-Process Analysis of Computer Integrated Manufacturing Documentation and Inspection Functions. *International Journal of Computer Integrated Manufacturing* 9(5), 339-353.

Dori, D. (1996b). Analysis and Representation of the Image Understanding *Environment Using the Object-Process Methodology*. Journal of Object Oriented Programming, 9(4), 30-38.

Dori, D. (2000). *Object-Process Methodology: A Holistic Systems Development Paradigm*. Springer Verlag (in press).

Dori, D. and Dori, Y.J. (1996). Object-Process Analysis of a Hypertext Organic Chemistry Studyware, Journal of Computers in Mathematics and Science Teaching, 15(1/2), 65-84.

Dori D. and Sturm, A. (1998). OPCAT - Object-Process CASE Tool - an Integrated System Engineering Environment (ISEE) OOPSLA'98 - Object-Oriented Programming, Systems, Languages

and Applications. Vancouver, BC, Canada, 18-22 October 1998. http://www.acm.org/sigplan/oopsla/oopsla98/fp/demos/demos.htm

Fitzgerald, B. (1997). The Use of Systems Development Methodologies in Practice: a field study. *Information Systems Journal* 7(3), 201-212.

Fowler, M. (1999). UML Distilled, Second Edition, Addison Wesley, Reading, MA.

Hall, Upper Saddle River, New Jersey.

Hanseth, O. and Monteiro E. *Understanding Information Infrastructure*. http://www.ifi.uio.no/~oleha/Publications/bok.html

Jarzabek, S. and Huang, R. (1998). The Case for User-Centered CASE Tools. *Communications of the ACM*, 41(8), 93-99.

Kappel, G. (1995). The Advocatus Diaboli of Object-Oriented Development, Dagstuhl Seminar Report 9434, 19.

Kovitz, B.L. (1998). Practical Software Requirements: A Manual of Content and Style, Manning Publications Company.

Lamond, K. (1996). Credit Card Taransactions: Real World and Online. http://www.virtualschool.edu/mon/ElectronicProperty/klamond/Overvw.htm

Latour, B. and Woolgar, S. (1979). *Laboratory Life: the Social Construction of Scientific Facts*. Sage, Beverly Hills and London.

Meyersdorf, D. and Dori, D. (1997). The R&D Universe and Its Feedback Cycles: an Object-Process Analysis. *R&D Management*, 27(4), 333-344.

OMG (2000) UML 1.3 Documentation: http://www.rational.com/uml/resources/documentation/index.jtmpl

Optner,Stanford L. (1973). *Systems Analysis* Penguin, Harmondsworth.

Peleg, M. and Dori, D. (1998). Representing Control Flow Constructs in Object-Process Diagrams, *Journal of Object-Oriented Programming*, 11(3), 58-71.

Peleg, M. and Dori, D. (1999). Extending the Object-Process Methodology to Handle Real-Time Systems. *Journal of Object-Oriented Programming* 11(8), 53-58.

Peleg, M. and Dori, D. (2000). The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. *IEEE Transaction on Software Engineering*, 26(7), 1-18 (to appear).

Rosenberg, J. M.(1993). : *Dictionary of Banking*, John Wiley & Sons Inc., New York.

Rumbaugh, J. , Blaha, M. , Premerlani, W., Eddy, F. and Lorenson, W. (1991). Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ.

Textcor (1998). *Electronic Commerce Primer*, http://www.textor.com/commerce/ecom.html

Wand, Y. and Wang, R.Y. (1996).Anchoring Data Quality Dimensions in Ontological Foundations, *Communications of the ACM*, 39(11), 86-95.

Underwood, J. (1998). Not Another Methodology: what ant tells us about systems development. *Proc. 6th International Conference on Information Systems Methodologies, British Computer Society*, Salford UK. http://www-staff.mcs.uts.edu.au/~jim/papers/ismeth.htm

Wenyin, L. and Dori, D.(1998). A Generic Integrated Line Detection Algorithm and its Object-Process Specification. *Computer Vision - Image Understanding*, 70(3), 420-437.

Wenyin, L. and Dori, D. (1999). Object-Process Diagrams as an Explicit Algorithm Specification Tool. *Journal of Object-Oriented Programming*, 12(2), 52-59.

Dov Dori is Visiting Associate Professor at the Engineering Systems Division at MIT, Cambridge, MA. He is Associate Professor at the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology and Founder and CEO of Systemantica, Inc., which develops OPM-based systems development environment. Dov Dori received his B.Sc. in Industrial Engineering and Management from the Technion in 1975, M.Sc. in Operations Research from Tel Aviv University in 1981, and Ph.D. in Computer Science from Weizmann Institute of Science, Rehovot, Israel, in 1988. His research interests include Systems Development Methodologies, Information Systems Engineering, Computer-Aided Software Engineering and Document Analysis and Recognition. Dov Dori is Associate Editor of IEEE Transaction on Pattern Analysis and Machine Intelligence (T-PAMI) and International Journal of Document Analysis and Recognition. He is Fellow of the International Association for Pattern Recognition (IAPR) Senior Member of IEEE, and Member of IEEE Computer Society and ACM.