Developing Complex Systems with Object-Process Methodology Using OPCAT

Dov Dori, Iris Reinhartz-Berger, and Arnon Sturm

Technion, Israel Institute of Technology
Technion City, Haifa 32000, Israel
Emails: {dori@ie, ieiris@tx, sturm@tx}.technion.ac.il

Abstract. OPCAT – Object-Process CASE Tool – is an Integrated Systems Engineering Environment which supports system development, lifecycle, and evolution using Object-Process Methodology (OPM). OPM integrates object- and process-oriented paradigms into a single frame of reference that provides for modeling the system in a single bimodal, graphic and textual model. This paper describes and demonstrates highlights of OPCAT and its unique capabilities, which include natural language generation, automated documentation and code generation, and animated simulation. Using a formal OPM metamodel, we present OPM-based system lifecycle development and how OPCAT serves this purpose.

1 Introduction

A system modeling and development methodology ideally supports the entire system lifecycle, from initiation (conceiving, initiating, and requirement elicitation) through development (analysis, design, and implementation) to deployment (assimilation, usage, and maintenance) [1]. To enable this diversified set of activities, the methodology should be based on sound ontology, which can be either universal or domain-specific [13]; a language for expressing the ontology; and a well-defined system development process. Developers who follow this process use the language to produce the artifacts that are pertinent for each phase of the system's lifecycle. It should therefore come as no surprise that any system modeling and development methodology worthy of its name is itself a highly complex system, and as such, it ought to be carefully analyzed and modeled.

The concept of metadata is quite widespread. In the context of the Internet, for example, metadata is machine understandable information for the Web. Metamodeling, the process of modeling a methodology, extends the notion of metadata and produces metamodels [5, 6], i.e., models of methodologies. Metamodels have become important means for building, comparing, and evaluating methodologies and their supporting CASE tools. Hence, it has been the focal point in several efforts to coalesce object-oriented methods and, at the same time, put them on a more rigorous footing. Some of the created metamodels use the methodology being modeled as a tool for describing itself. We refer to this type of metamodeling as **reflective metamodeling** and to the methodology as a **reflective methodology**. A reflective methodology is especially powerful since it is self-contained and does not require auxiliary means or external tools to model itself.

Most of the existing (both reflective and non-reflective) metamodels focus on describing the syntax and semantics of the methodology constructs, leaving out of the metamodel all the procedural and behavioral aspects. These aspects relate to processes that are either part of the language capabilities (such as refinement-abstraction processes) or processes that belong to the development of a system using the methodology. The reason for the lack of procedural modeling is that the techniques used for metamodeling (such as ERD and UML) are structural- or object-oriented. Object-Process Methodology (OPM) overcomes this limitation by supporting the specification of the structural and behavioral aspects of the modeled methodology in a single framework, enabling mutual effects between them.

In this paper, we apply OPM to define a comprehensive lifecycle-supporting system development process. This process follows generic concepts of systems evolution and lifecycle, namely requirement specification, analysis and design, implementation, usage and maintenance, and, as such, it is not specific to OPM-based system development. Nevertheless, applying it in an OPM framework has great benefits. In Section 2 we review existing metamodels and criticize their ability to model system development processes. In Section 3 we introduce the foundations of OPM, while the metamodel of an OPM-based development process is presented in Section 4. Finally, in Section 5, we summarize the main benefits of our metamodeling approach and discuss future research directions.

1 Object-Process Methodology Highlights

Object-Process Methodology (OPM) [1] is a holistic approach to the study and development of systems. It integrates the object-oriented and process-oriented paradigms into a single frame of reference. Structure and behavior, the two major aspects that each system exhibits, co-exist in the same OPM view, expressed both graphically and in natural language, without highlighting one at the expense of suppressing the other. The elements of the OPM ontology, presented in Appendix A, are entities (things and states) and links. A thing is a generalization of an object and a process. Objects are (physical or informatical) things that exist, while processes are things that transform objects. Links can be structural or procedural. Structural links express static relations between pairs of entities. Procedural links connect entities to describe the behavior of a system. The behavior is manifested in three major ways: processes can transform objects, objects can enable processes, and objects can trigger events that invoke processes. Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related 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 by which entities can be linked. The Object-Process Language (OPL), defined by a grammar, is the textual counterpart modality 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 analyzing and designing a system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase. Designed also for machine interpretation, OPL provides a solid basis for automatically generating the designed application. This dual representation of OPM increases the processing capability of humans.

A modeling and development methodology is a combination of a language for expressing the universal or domain ontology and an approach for developing systems using that language. A common way for building, comparing, and evaluating methodologies is metamodeling, i.e., the process of modeling the methodology. Most of the methodology metamodels pertain only to the language part of the methodology, leaving out the description of the system development processes or describing them informally. A major reason for this is that the methods used for metamodeling are structure- or object-oriented, and, hence, are less expressive in modeling the procedural aspects of a methodology. In this paper we apply Object-Process Methodology (OPM) to specify a generic OPM-based system development process. This metamodel is made possible due to OPM's view of objects and processes as being on equal footing rather than viewing object classes as superiors to and owners of processes. This way, OPM enables specifying both the structural (ontological constructs) and behavioral (system development) aspects of a methodology in a single, unified view.

3 Metamodels and Metamodeling

System analysis and design activities can be divided into three types with increasing abstraction levels: real world, model, and metamodel. The real world is what system analysts perceive as reality or what system architects wish to create as reality. A model is an abstraction of this perceived or contemplated reality that enables its expression using some approach, language, or methodology. A metamodel is a model of a model, or more accurately, a model of the modeling methodology. Analogous to modeling, metamodeling is the process that creates metamodels. The level of abstraction at which metamodeling is carried out is higher than the level at which modeling is normally done for the purpose of generating a model of a system.

The growth of object-oriented methods during the last decade of the 20^{th} century introduced a special type of metamodeling, which we call **reflective metamodeling**. Reflective metamodeling models a methodology by the means and tools that the methodology itself provides. While metamodeling is a formal definition technique of methodologies, reflective metamodeling can serve as a common way to examine and demonstrate the methodology’s expressive power.

Metamodels of visual software engineering methods are commonly expressed in ER or class diagrams. These notations model primarily the structural and static aspects of methodologies. ER-based metamodels are also limited in describing constraints, hierarchical structures (i.e., complex objects), explosion, and polymorphism required for specifying complete methodologies or languages.

UML [9, 14], which is the standard object-oriented modeling language, has several metamodel propositions. The reflective UML metamodel in [9], for example, includes class diagrams, OCL (Object Constraint Language) sentences, and natural language explanations for describing the main elements in UML and the static relations among them. The Meta Object Facility (MOF) [7], which is an OMG standard, extensible four layer metadata architecture, is also applied to metamodel UML. MOF layers are: information (i.e., real world concepts, labeled M0), model (M1), metamodel (M2), and meta-metamodel (M3). The meta-metamodel layer describes the structure and semantics of meta-metadata, i.e., it is an "abstract language" for defining different kinds of metadata (e.g., meta-classes and meta-attributes). The Meta Modeling Facility (MMF) [7] provides a modular and extensible method for defining and using UML. It comprises a static, object-oriented language (MML), used to write language definitions; a tool (MMT) used to interpret those definitions; and a method (MMM), which provides guidelines and patterns encoded as packages that can be specialized to particular language definitions.

These metamodels of UML are incomplete in more than one way. First, UML is only a language, not a methodology, so only the language elements are metamodeled, but not any object-oriented (or other) development process [9]. Second, the consistency and integrity constraints that UML models should follow are not included and formulated in these metamodels. Several "software process models" have been associated with UML to create complete UML-based methods. One such familiar development process is the Rational Unified Process (RUP) [12]. RUP is a configurable software development process pattern that presents the relations between the process lifecycle aspects (inception, elaboration, construction, and transition) and the process disciplines and activities (business modeling, requirements, etc.). While RUP supplies a general framework of development processes, it does not have a precise underlying metamodel.

The Software Process Engineering Metamodel (SPEM) [8] uses UML to describe a concrete software development process or a family of related software development processes. It uses MOF four-layered architecture, where the performing process (the real-world production process) is at level M0 and the definition of the corresponding process (e.g., RUP) is at level M1.

The Object-oriented Process, Environment, and Notation (OPEN) [3, 4, 100] is a methodology that offers a notation, called OPEN Modeling Language (OML), as well as a set of principles for modeling all aspects of software development across the entire system lifecycle. The development process is described by a contract-driven lifecycle model, which is complemented by a set of techniques and a formal representation using OML. The lifecycle process, including its techniques, tasks, and tools, is described in terms of classes and their structural relations.

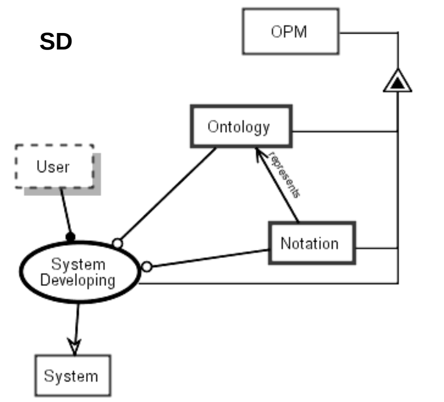
The above metamodels, as well as other metamodels that use structural- or object-oriented methodologies, emphasize the objects and their relations within the metamodel, while the procedural aspects are suppressed and revealed only through operations of objects and the messages passed among them. While real-world processes require interaction and state diagrams to describe system dynamics and function, metamodels of methodologies use only packages, classes, and associations. The main reasons for this limited usage of UML include the complexity of its vocabulary and its model multiplicity and integration problems [11]. Object-Process Methodology overcomes this shortcoming by recognizing processes as entities beside, rather than underneath, objects.

4 An OPM-Based System Development Model

The System Diagram, which is labeled **SD** and shown in Figure 1, is the top-level specification of the OPM metamodel. It specifies **Ontology**, **Notation,** and the **System Developing** process as the major OPM features (characterizations). **Ontology** includes the basic elements in OPM, their attributes, and the relations among them. For example, objects, processes, states, and aggregations are all OPM elements. The **Notation** represents the **Ontology** graphically (by OPDs) or textually (by OPL sentences). For example, a process is represented graphically in an OPD by an ellipse, while an object is symbolized by a rectangle.

The **System Developing** process, also shown in **SD**, is handled by the **User**, who is the physical and external (environmental) object that controls (is the agent of) the process. This process also requires **Ontology** and **Notation** as instruments (inputs) in order to create a **System**.

The OPL paragraph, which is equivalent to SD, is also shown in Figure 1. Since OPL is a subset of English, users who are not familiar with the graphic notation of OPM can validate their specifications by inspecting the OPL sentences. These sentences are automatically generated on the fly in response to the user's draws of OPDs [2]. Due to space limitations and the equivalence of OPM graphical and textual notations, we use only the OPD notation in the rest of the paper.



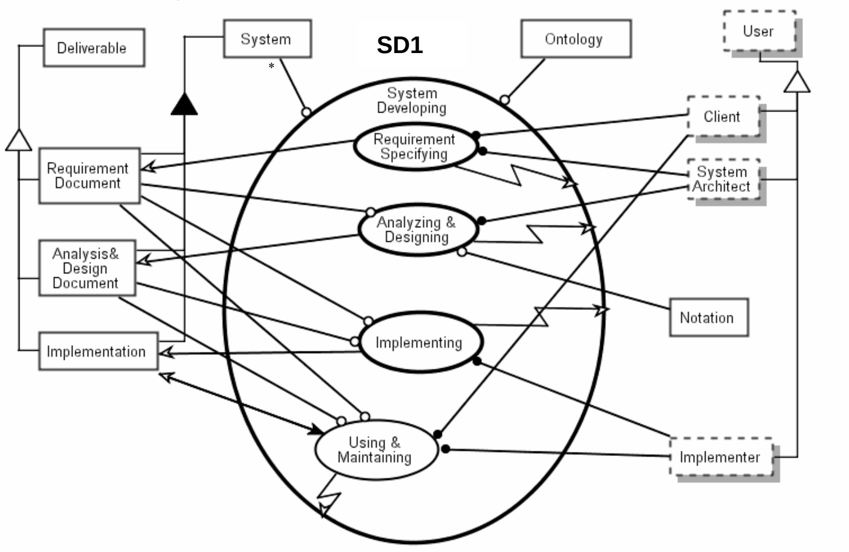
OPM exhibits **Ontology** and **Notation**, as well as **System Developing.**
Notation represents Ontology.
System Developing requires **Notation** and **Ontology.**
System Developing yields **System.**
User is environmental and physical**.**
User handles **System Developing.**

Figure 1. The top level specification of the OPM metamodel

Zooming into System Developing, SD1 (Figure 2) shows the common sequential[1] stages of system developing processes: Requirement Specifying, Analyzing & Designing, Implementing, and Using & Maintaining. All of these processes use the same OPM Ontology, a fact that helps narrowing the gaps between the different stages of the development process. SD1 shows that the Client and the System Architect, which, along with the Implementer, specialize User, handle the Requirement Specifying sub-process. Requirement Specifying takes OPM Ontology as an input and creates a new System, which, at this point, consists only of a Requirement Document. The termination of Requirement Specifying starts Analyzing & Designing, the next sub-process of System Developing.

The agent of the Analyzing & Designing stage is the System Architect, who uses the Requirement Document and OPM Notation to create a new part of the system, the Analysis & Design Document. When the Analyzing & Designing process terminates, the Implementer (programmer, DBA, etc.) starts the Implementing phase, which uses the Requirement Document and the Analysis & Design Document in order to create the Implementation. Finally, the Implementer changes the system Implementation during the Using & Maintaining stage, while the Client uses the System.

As the invocation links in SD1 denote, each System Developing sub-process can invoke restarting of the entire development process, which potentially enables the introduction of changes to the requirements, analysis, design, and implementation of the System. These invocations give rise to an iterative development process, in which an attempt to carry out a sub-process reveals faults in the deliverable of a previous subprocess, mandating a corrective action.



[1] The time line in an OPD flows from the top of the diagram downwards, so the vertical axis within an in-zoomed process defines the execution order. The sub-processes of a sequential process are depicted in the in-zoomed ellipse of the process stacked on top of each other with the earlier process on top of a later one. Analogously, subprocesses of a parallel process appear in the OPD side by side, at the same height.

Figure 2. Zooming into System Developing

4.1 The Requirement Specifying stage

In SD1.1 (Figure 3), Requirement Specifying is zoomed into, showing its four subprocesses. First, the System Architect and the Client define the problem to be solved by the system (or project). This Problem Defining step creates the Problem Definition part of the current system Requirement Document. Next, through the Requirement Reusing sub-process, the System Architect may reuse requirements that fit the problem at hand and are adapted from any existing System (developed by the organization). Reuse helps achieve high quality systems and reduce their development and debugging time. Hence, when developing large systems, such as Web applications or real-time systems, it is important to try first to reuse existing artifacts adapted from previous generations, analogous systems, or commercial off-the-shelf (COTS) products that fits the current system development project. Existing, well-phrased requirements are often not trivial to obtain, so existing relevant requirements should be treated as a potential resource no less than code. Indeed, as the OPD shows, reusable artifacts include not only components (which traditionally have been the primary target for reuse), but also requirements.

After optional reuse of requirements from existing systems (or projects), the System Architect and the Client, working as a team, add new Requirements or update existing ones. This step uses OPM Ontology in order to make the Requirement Document amenable to be processed by other potential OPM tools, and in particular to an OPL compiler. The bi-modal property of OPM, and especially the use of OPL, a subset of natural language, enables the Client to be actively involved in the critical Requirement Specifying stage. Moreover, since the System Architect and the Client use OPM Ontology in defining the new requirements, the resulting Requirement Document is indeed expressed, at least partially, in OPL in addition to explanations in free natural English. Such structured OPM-oriented specification enables automatic translation of the Requirement Document to an OPM analysis and design skeleton (i.e., a skeleton of an OPD-set and its corresponding OPL script). Naturally, at this stage the use of free natural language beside OPM seems mandatory to document motivation, alternatives, considerations, etc.

Finally, the Requirement Adding process results in the Boolean object “Is Backtracking Required?”, which determines whether System Developing should be restarted. If so, Development Process Backtracking invokes the entire System Developing. Otherwise, Requirement Specifying terminates, enabling the Analyzing & Designing process to begin.

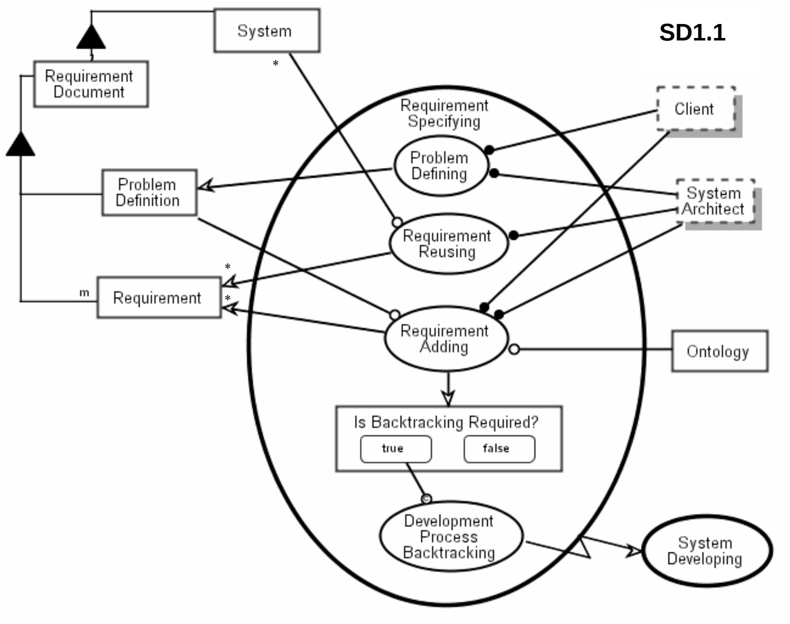


Figure 3. Zooming into Requirement Specifying

4.2 The Analyzing and Designing stage

During the Analyzing & Designing stage, shown in SD1.2 (Figure 4), a skeleton of an OPL Script is created from the Requirement Document for the current system. As noted, in order to make this stage as effective and as automatic as possible, the Requirement Document should be written using OPM, such that the resulting OPL script can be compiled. The System Architect can then optionally reuse analysis and design artifacts from previous systems (projects), creating a basis for the current system analysis and design. Finally, in an iterative process of Analysis & Design Improving (which is in-zoomed in SD1.2.1, Figure 5), the System Architect can engage in OPL Updating, OPD Updating, System Animating, General Information Updating, or Analysis & Design Terminating.

SD1.2

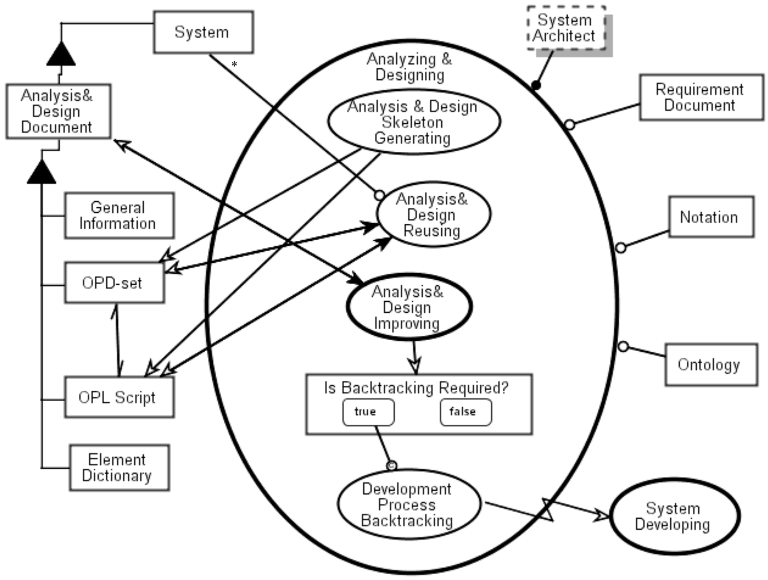


Figure 4. Zooming into Analyzing & Designing

Any change a user makes to one of the modalities representing the model triggers an automatic response of the development environment software to reflect the change in the complementary modality. Thus, as SD1.2.1 shows, OPD Updating (by the System Architect) affects the OPD-set and immediately invokes OPL Generating, which changes OPL Script according to the new OPD-set. Conversely, OPL Updating (also by the System Architect) affects the OPL Script, which invokes OPD Generating, reflecting the OPL changes in the OPD-set.

Since OPM enables modeling system dynamics and control structures, such as events, conditions, branching, and loops, **System Animating** simulates an **OPD-set**, enabling **System Architects** to dynamically examine the system at any stage of its development. Presenting live animated demonstrations of system behavior reduces the number of design errors percolated to the implementation phase. Both static and dynamic testing help detecting discrepancies, inconsistencies, and deviations from the intended goal of the system. As part of the dynamic testing, the simulation enables designers to track each of the system scenarios before writing a single line of code. Any detected mistake or omission is corrected at the model level, saving costly time and efforts required within the implementation level. Avoiding and eliminating design errors as early as possible in the system development process and keeping the documentation up-to-date contribute to shortening the system's delivery time ("time-to-market").

Upon termination of the Analysis & Design Improving stage, if needed, the entire System Developing process can restart or the Implementing stage begins.

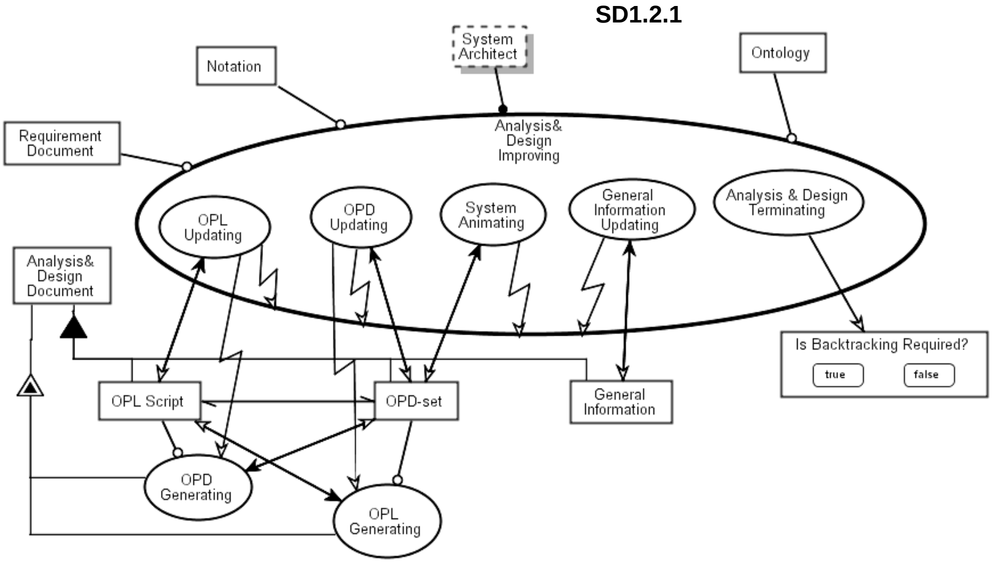


Figure 5. Zooming into Analysis & Design Improving

4.3 The Implementing stage

The Implementing stage, in-zoomed in SD1.3 (Figure 6), begins by defining the Implementation Profile, which includes the target Language (e.g., Java, C++, or SQL) and a default Directory for the artifacts. Then, the Implementation Skeleton Generating process uses the OPL Script of the current system and inner Generation Rules in order to create a skeleton of the Implementation. A Generation Rule saves pairs of OPL sentence types (templates) and their associated code templates in various target Languages.

The initial skeleton of the Implementation, which includes both the structural and behavioral aspects of the system, is then modified by the Implementer during the Implementation Reusing and Implementation Improving steps. In the Testing & Debugging stage, the resulting Implementation is checked against the Requirement Document in order to verify that it meets the system requirements defined jointly by the Client and the System Architect. If any discrepancy or error is detected, the System Developing process is restarted, else the system is finally delivered, assimilated and used. These sub-processes are embedded in the Using & Maintaining process at the bottom of SD1 (Figure 2). While Using & Maintaining takes place, the Client collects new requirements that are eventually used when the next generation of the system is initiated. A built-in mechanism for recording new requirements in OPM format while using the system would greatly facilitate the evolution of the next system generation [1].

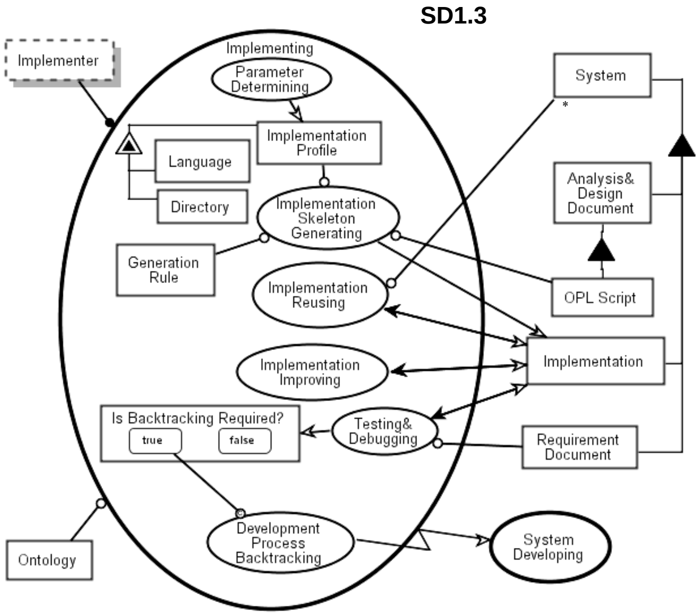


Figure 6. Zooming into Implementing

5 Summary and Future Work

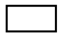
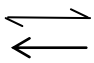
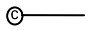
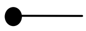
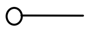
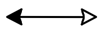
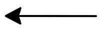
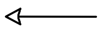
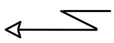
We have presented a complete and detailed model of a system for developing systems as part of the OPM reflective metamodel. This system development model follows generic concepts of systems evolution and lifecycle, and as such, it is not specific to OPM-based system development. Nevertheless, applying this process in an OPM framework has great benefits: it narrows the gap between the various development steps and enables semi-automated generations. The elaborate backtracking options of this model, which are built-in at all levels, make it flexible enough to represent a variety of information system development approaches, ranging form the classical waterfall model through incremental development to prototyping.

Although object-oriented system development methods have been augmented to include models that enable specification of the system's behavioral aspects (e.g., UML sequence, collaboration, and Statechart diagrams), formal metamodels of these methods relate only to their language aspects. More specifically, the widely accepted object-oriented approach, which combines UML as the language part with RUP as the system development part, provides a formal metamodel only of the static aspects. Conversely, since OPM inherently combines the system's structural and behavioral aspects in a unifying, balanced framework, it can reflectively metamodel both the language and the development process parts of any methodology. This ability to model equally well structural and procedural system aspects is indicative of OPM's expressive power, which is a direct result of its balanced ontology. Recognizing objects and processes as prime ontological constructs of equal status provides for faithful modeling of systems, regardless of their domain, while OPM's abstraction-refinement capabilities enable systems' complexity management.

The system development process specified in this work is designed to accompany the development of any system that involves a combination of complex structure and behavior. The model of this development process provides a theoretical foundation for improving the current version of OPCAT [2], Object Process CASE Tool, that supports OPM-based systems development. **System Animating**, **OPD Updating**, and **OPL Updating** are already implemented as OPCAT services, while **Implementation Skeleton Generating** is in progress. We already have a generic code generator and We also plan to implement and incorporate all the other **System Developing** sub-processes into OPCAT in order to make it a fully Integrated System Engineering Environment (I SEE).

Appendix A

Main OPM elements, their symbols, and their semantics

Element	Symbol	Semantics
Informatical object		A piece of information
Environmental, physical object		An object which consists of matter and/or energy and is external to the system
Process class		A pattern of transformation that objects undergo
State		A situation at which an object can exist for a period of time
Characterization		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
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

References

1 Dori, D. Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002.

2 Dori, D. Reinhartz-Berger, I. and Sturm A. *OPCAT* – A Bimodal Case Tool for Object-Process Based System Development. 5th International Conference on Enterprise Information Systems (ICEIS 2003), pp. 286-291, 2003. Software download site: http://www.objectprocess.org

3 Firesmith, D., Henderson-Sellers, B., and Graham, I. The OPEN Modeling Language (OML) – Reference Manual. Cambridge University Press, SIGS books, 1998.

4 Graham, I., Henderson-Sellers, B., and Younessi, H. The OPEN Process Specification. Addison-Wesley Inc., 1997.

5 Henderson-Sellers, B. and Bulthuis, A. Object-Oriented Metamethods, Springer Inc., 1998.

6 Hillegersberg, J.V., Kumar, K. and Welke, R.J. Using Metamodeling to Analyze the Fit of Object-Oriented Methods to Languages. Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS'98), pp. 323-332, 1998.

7 Object Management Group (OMG). Meta Object Facility (MOF) Specification. OMG document formal/02-04-03, http://cgi.omg.org/docs/formal/02-04-03.pdf

8 Object Management Group (OMG). Software Process Engineering Metamodel (SPEM), version 1.0, OMG document formal/02-11-14, http://www.omg.org/technology/documents/formal/spem.htm

9 Object Management Group (OMG). UML 1.4 - UML Semantics. OMG document formal/01-09-73, http://cgi.omg.org/docs/formal/01-09-73.pdf

10 OPEN web site, http://www.open.org.au/

11 Peleg, M. and Dori, D. The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transaction on Software Engineering, 26 (8), pp. 742-759, 2000.

12 Rational Software. Rational Unified Process for Systems Engineering – RUP SE1.1. A Rational Software White Paper, TP 165A, 5/02, 2001, http://www.rational.com/media/whitepapers/TP165.pdf

13 Talvanen, J. P. Domain Specific Modelling: Get your Products out 10 Times Faster. Real-Time & Embedded Computing Conference, 2002, http://www.metacase.com/papers/Domain-specific_modelling_10X_faster_than_UML.pdf

14 Siau, K. and Cao, Q. Unified Modeling Language (UML) – A Complexity Analysis. Journal of Database Management 12 (1), pp. 26-34, 2001.

15 Van Gigch, J. P. System Design Modeling and Metamodeling. Plenum press, 1991.

16 Warmer, J. and Kleppe, A. The Object Constraint Language – Precise Modeling with UML. Addison-Wesley, 1999.