Evaluating Aspects of Systems Modeling Languages by Example: SysML and OPM

Yariv Grobshtein and Dov Dori
Faculty of Industrial Engineering and Management
Technion – Israel Institute of Technology
Technion City, Haifa 32000
Israel
{yarivg@tx, dori@ie}.technion.ac.il

Abstract. As systems are becoming larger and more complex, the challenge of developing quality systems efficiently is on the rise. While traditionally document-centric approaches have been used, in recent years the benefits of model-based systems engineering have been acknowledged. Recognizing the importance of modeling as a key factor in managing system development complexity, the selection of the modeling language to be used should be considered. This work evaluates aspects of two systems modeling languages: SysML – OMG's Systems Modeling Language and OPM – Object-Process Methodology. The evaluation was done on the basis of a concrete sample problem, in which multiple aspects of the system were modeled in both SysML and OPM. Some of the findings, which were generalized from the case study, suggest that OPM is usually advantageous in presenting the system different hierarchy levels and combining structure with behavior, while SysML is more convenient for modeling detailed views of some aspects.

Introduction

One of the main goals of conceptual modeling is to improve system analysis and enable its architecting and detailed design. Early model construction provides for verification and validation of a system prior to implementation and integration of its components. Other benefits of a conceptual model include improving the communication between the different stakeholders and enabling the reuse of some of its components. A complete model should support various aspects of a system, notably the three fundamental ones—the functional, behavioral, and structural aspects. Function pertains to the goal the system is designed for, while the combination of structure and behavior, commonly called the system's architecture, embodies the concept upon which the solution of the function execution problem is based.

Several modeling languages have been designed for general-purpose systems, each with its own semantics and notation. Typically, a Systems Modeling Language (SML) provides tools for representing a system in visual and/or textual modalities. Choosing the appropriate SML may have large impact on the success of the model construction, hence it is important and interesting to evaluate the effectiveness of different SMLs.

This work focuses on two of the state-of-the-art representative SMLs: OMG Systems Modeling Language (SysML) (OMG 2007c, OMG 2007d) and OPM – Object-Process Methodology (Dori 2002). SysML is based on UML 2 (OMG 2007a, OMG 2007b), which is widely used as a de-facto standard in software engineering, with several extensions and

modifications for general systems engineering. In contrast, OPM was initially designed to support modeling of general-purpose systems, thus it has no inherent “software oriented” language semantics. Furthermore, OPM treats software systems as specializations of general systems. For example, OPM objects and processes can be physical, which is typical of systems in general, or informatical, which exist in models of both software systems and general systems. Another major difference is the number of views (diagram types) used in each language. While OPM is based on a single diagram type—Object-Process Diagram (OPD), SysML has inherited UML's model multiplicity (Peleg and Dori 2000), i.e., it presents each one of the system's aspects in a different view using a different diagram type. SysML includes several types of UML diagrams, as well as two new types of diagrams for systems engineering, Requirement Diagram and Parametric Diagram. OPD, OPM's single type of diagram, is missing some elements that are important for systems engineering, such as the SysML parametric constraints. Both languages support hierarchical representation of the model. However, in contrast to SysML, where the model is represented in separate views with partial support of hierarchy, in OPM, the entire system model is based on a well-defined hierarchy of OPDs. These are but few of several dissimilarities between the languages, which make it interesting to learn and compare them.

Extending the work of Grobshtein et al. (2007), we have selected a Hybrid gas/electric powered Sport Utility Vehicle (SUV), taken from the SysML specification (OMG 2007c) as a sample system to support this investigation. Various aspects of this system have been modeled in both SysML and OPM. Several aspects of the reference Hybrid SUV SysML model were matched with a semantically equivalent OPM model in order to compare and characterize the two languages. Analysis and generalization of the observations from the particular example are then provided. Conclusions and future work are finally suggested.

Overview of OPM and SysML

As a prelude to our comparative investigation, we briefly describe the two languages considered in this work. The description of OPM is more extensive, as it is less familiar than UML and SysML.

Object-Process Methodology

Object-Process Methodology (OPM) is a holistic approach to conceptual modeling of complex systems. The OPM model integrates structural, functional and behavioral aspects of a system in a single, unified view, expressed bi-modally in equivalent graphics and text with built-in refinement-abstraction mechanism.

The graphic modality is expressed via a set of Object-Process Diagrams (OPDs) and the textual—via Object-Process Language (OPL). OPDs include both the entities of the model (objects, processes, and states) and links and relations among them, as well as data to preserve the graphical representation of the model elements (size, location, etc.). OPL equivalently specifies the same OPM model in a subset of English, enabling one-to-one mapping between the graphic and the textual representations, such that two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related, hierarchically organized OPDs show portions of the system at various levels of detail. The OPM ontology comprises entities and links. Each OPM element (entity or link) is denoted in an OPD by a symbol, and the OPD syntax specifies correct and consistent ways by which entities can be connected via structural and procedural links, such that each legal entity-link-entity combination bears specific, unambiguous semantics. See Figure 1 for example.

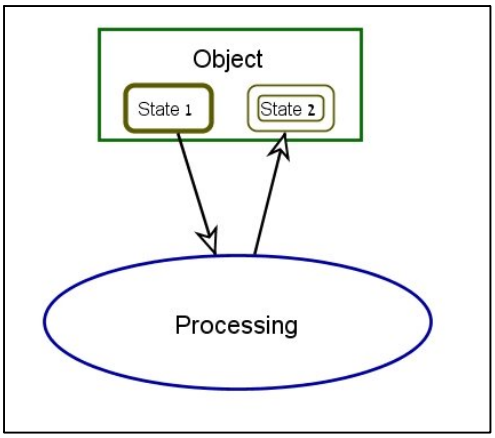

Figure 1. An Object-Process Diagram (OPD) showing the three OPM entities: Object, Process, and State, and the input/output procedural link pair, which expresses that Processing changes Object from State 1 to State 2.

There are three different types of entities: objects, processes (collectively referred to as "things"), and states. These entities are shown in Figure 1. Objects are the (physical or informatical) things in the system that exist, and if they are stateful (i.e., have states), then at any point in time they are at some state or in transition between states. Processes are the things in the system that transform objects: they generate and consume objects, or affect stateful objects by changing their state.

Links can be structural or procedural. Structural links express static, time-independent relations between pairs of entities. The four fundamental structural relations are aggregation-participation, generalization-specialization, exhibition-characterization, and classification-instantiation. General tagged structural links provide for creating additional "user-defined" links with specified semantics, similar to association links in SysML block diagrams. Procedural links connect processes with objects or object states to describe the behavior of a system. System behavior is manifested in three ways: (1) A processes can transform (generate, consume, or change the state of) one or more objects; (2) an object can enable one or more processes without being transformed by them, in which case it acts as an agent (if it is human) or an instrument; and (3) an object can trigger an event that invokes a process if some conditions are met. Accordingly, a procedural link can be a transformation link, an enabling link, or an event link. A transformation link expresses object transformation, i.e., object consumption, generation, or state change. Figure 1 shows a pair of transformation links, the input and output links. It expresses in OPL that "Processing changes Object from State 1 to State 2." An enabling (agent or instrument) link expresses the need for a (possibly state-specified) object to be present in order for the enabled process to occur. The enabled process does not transform the enabling object. An event link connects a triggering entity (object, process, or state) with a process that it invokes.

The System Diagram, SD, is the topmost diagram in a model. It presents the most abstract view of the system, typically showing a single process as the main function of the system, along with the most significant objects that enable it and the ones that are transformed by it. The further from the root the OPD is, the more detailed it is. Each OPD, except for the SD, is obtained by refinement – in-zooming or unfolding – of a thing (object or process) in its ancestor OPD. This refined thing is described with additional details. The abstraction-refinement mechanism ensures that the context of a thing at any detail level is never lost and the "big picture" is maintained at all times.

A new thing (object or process) can be presented in any OPD as a refineable (part, specialization, feature, or instance) of a thing at a higher abstraction level (a more abstract OPD). It is sufficient that some detail appears once in some OPD in order for it to be true for the system in general. Accordingly, copies of a thing can appear in other diagrams, where some or all the details (such as object states or thing relations) that are unimportant in the context of a particular diagram need not be shown.

The Object-Process Language (OPL) 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 subset of English, which serves domain experts and system architects, jointly engaged in modeling a complex system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase that is generated automatically by an OPM-supporting modeling tool, such as OPCAT (Dori et al. 2003), in response to the user's input. According to the modality principle of the cognitive theory of multimodal learning (Mayer 2001), this dual graphic/text representation of the OPM model increases the human processing capability. It has indeed been our experience that humans enhance their understanding of the model as they conveniently draw upon the graphic and/or the textual model representation to complement what they missed in the other modality.

A major problem with most graphic modeling approaches is their scalability: As the system complexity increases, the graphic model becomes cluttered with symbols and links that connect them. The limited channel capacity (Mayer 2001) is a cognitive principle which states that there is an upper limit on the amount of detail a human can process before being overwhelmed. This principle is addressed by OPM and implemented in OPCAT with three abstraction/refinement mechanisms. These enable complexity management by providing for the creation of a set of interrelated OPDs (along with their corresponding OPL paragraphs), each of which is limited in size, thereby avoiding information overload and enabling comfortable human cognitive processing. The three refinement/abstraction mechanisms are: (1) unfolding/folding, which is used for refining/abstracting the structural hierarchy of a thing and is applied by default to objects; (2) in-zooming/out-zooming, which exposes/hides the inner details of a thing within its frame and is applied primarily to processes; and (3) state expressing/suppressing, which exposes/hides the states of an object.

OPCAT (Dori et al. 2003) is a software product that supports OPM-based system development and lifecycle management by implementing the hierarchical, bimodal expression of the OPM model. The OPCAT platform supports system requirements management, including interconnections and traceability to the model entities. Additional features include animated simulation of the model, code generation, and automatic document generation. The OPM model of the Hybrid SUV system, shown in the next section of this paper, was developed in OPCAT.

Systems Modeling Language

The drive to adapt UML to systems engineering applications brought about the establishment of the OMG Systems Engineering Domain Special Interest Group (SE DSIG). This OMG group, supported by the International Council on Systems Engineering (INCOSE) and ISO AP 233 workgroup, worked together on the requirements of the modeling language. The result was the UML for Systems Engineering RFP (OMG 2003) issued by the OMG in March 2003. SysML was the only response to the RFP. The SysML team consisted of industry users, tool vendors, government agencies, professional organizations and academia. Four and a half years after the RFP was published, version 1.0 of the SysML specification was formally released by OMG as an OMG specification in September 2007 (OMG 2007c).

A new general-purpose modeling language for systems engineering, SysML is intended to support specification, analysis, design, verification, and validation of complex systems. The systems may be of broad range, and can include hardware, software, data, personnel, procedures, facilities, and more.

SysML reuses a subset of UML 2 and provides additional extensions in order to satisfy the RFP requirements. As a visual modeling language, SysML offers several types of diagrams

which can reflect various aspects of a system. It is common to partition SysML diagrams into four "pillars" – structure, behavior, requirements, and parametric relationships. In addition, SysML provides means to cross-connect the different model elements. The tutorial of Friedenthal et al. (2007) contains examples of the various SysML diagram types.

Overall, SysML includes nine types of diagrams: four types of structure diagrams, four types of behavior diagrams, and a requirements diagram. In this section, we discuss mainly the SysML extensions and modifications with respect to UML, which make it more relevant to conceptual modeling of systems. SysML introduces two new diagram types: the Requirement Diagram and the Parametric Diagram. In addition, SysML modifies three types of diagrams from UML: Block Definition Diagram, Internal Block Diagram, and Activity Diagram. The remaining four SysML diagrams are reused from UML without new features.

Requirements: SysML supplies means to represent text-based requirements and to connect them to other model elements. A basic requirement is composed of a unique identifier and text properties. The requirements diagram can be shown in different formats – graphical, tabular, or tree structure. Requirements can also be part of other diagrams, reflecting relationship to other model constructs. Generally, the SysML requirements constructs are not meant to replace the external requirements management tools, but rather to better integrate the system requirements with other parts of the model.

The SysML specification provides several relationships among requirements, such as requirements hierarchies, source-derived requirement dependencies, satisfaction relations between requirements and the model, and verification dependencies to test-cases.

Structure: The basic structural element in SysML is Block. It can be used to describe physical or logical elements of the system, such as hardware, software, data, or persons. Blocks can describe any level of the system hierarchy, from single components up to the top-level system.

There are two types of structural diagrams for blocks depiction: Block Definition Diagram and Internal Block Diagram. The Block Definition Diagram describes the relationships among blocks, such as associations, dependencies, and generalizations. It specifies system hierarchy and classifications. The Internal Block Diagram represents the internal structure of a block using block properties and connectors between properties. This diagram specifies interconnection of parts. Another SysML structural diagram, the UML Package Diagram, is used to organize the model by grouping model elements.

Parametric: Parametric Diagram is a new diagram type that was introduced into SysML. By providing the ability to express constrains between properties, a Parametric Diagram enables integration of engineering analysis, such as performance and reliability models with SysML design models. The constraints (equations), including the underlying parameters, are captured in ConstraintBlock constructs. For example, a ConstraintBlock can have the parameters *F*, *m*, and *a*, and the constraint {*F=m*a*}. Constraint blocks are defined on a Block Definition Diagram, so they can be reused. Parametric Diagrams contain usage of constraint blocks to constrain the value properties of other blocks. This is done by binding the constraint parameters (such as *m*) to specific actual value properties of a block (such as the mass of a vehicle).

Behavior: SysML specifies four types of behavioral diagrams: Activity Diagram, Sequence Diagram, State Machine Diagram, and Use Case Diagram. The role of the Activity Diagram is to represent the flow of inputs and outputs and the flow of control between actions. It incorporates sequences and conditions for coordinating activities. Activities and activity diagrams exist also in UML, but SysML provides several extensions (Bock 2006), including means to support what

is known as "continuous" flow modeling, such as rate restrictions. Support for probabilities and extensions to control in activity diagrams (known as "Control as Data") were added.

The Hybrid Sport Utility Vehicle Case Study

Our approach to evaluating, comparing, and analyzing the two modeling languages under study leans on the investigation of a concrete modeling problem. To this end, we have selected the SysML specification sample problem from (OMG 2007c). The system to be modeled is a Hybrid gas/electric powered Sport Utility Vehicle (SUV). For each of the function, structure and behavior aspects of the problem, we bring the SysML diagrams used to model it from (OMG 2007c), followed by their semantically equivalent OPDs. We compare and analyze the different and similar points, and then try to generalize our observations.

This section is divided into three subsections. The first one deals with establishing the system context, system boundaries, and top-level use cases. The next subsection is dedicated to analysis and elaboration of top-level system behavior. The third subsection focuses on system structure, including hierarchy and relationships among components.

System Context, Boundaries and Use Cases

We start modeling the Hybrid SUV system by examining Figure 2, a context diagram which is a specific use of SysML Internal Block Diagram. It depicts some of the top-level entities in the overall enterprise and their relationships. In order to distinguish the system from its environment, the user-defined «system» and «external» stereotypes are used (applied to blocks). The "actor" symbol is also used to represent a model element which is part of the environment.

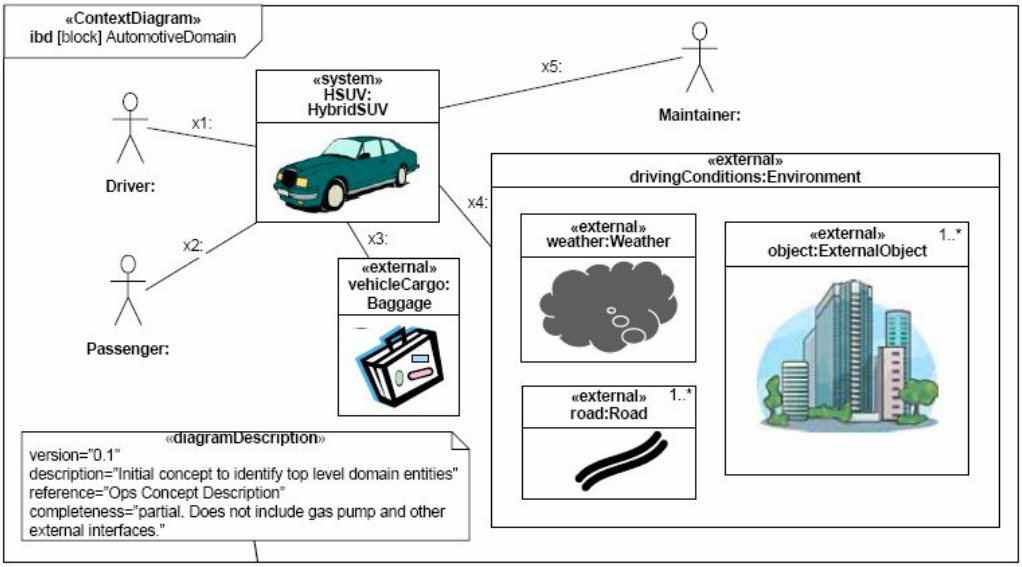

Figure 2. Context of the Hybrid SUV system in SysML

Figures 3 and 4 contain the OPDs that describe the same type of information in OPM. Figure 3 is the System Diagram (SD)—the top-level diagram in the OPD hierarchy, which includes several objects, as well as one process, called Vehicle Using. This process is the main function of the system, to be decomposed in lower-level OPDs. Vehicle Using is connected to the objects that enable it – in our case the Hybrid SUV and the various users (Driver, Passenger and Maintainer). Environmental (non-systemic) things are denoted by a dashed border, while physical things (as

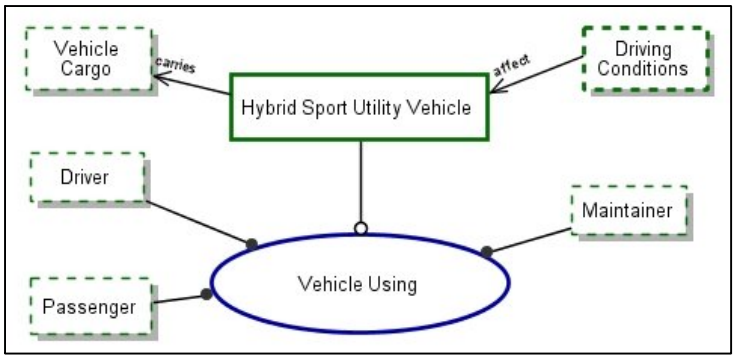

Figure 3. System Diagram (SD) of the Hybrid SUV system in OPM

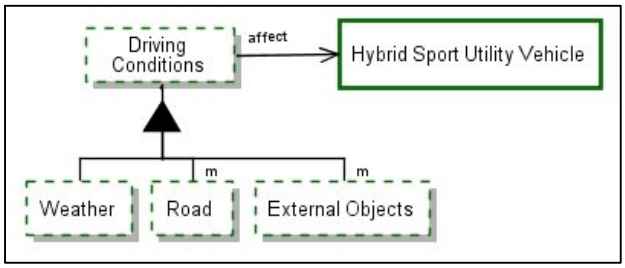

Figure 4. Driving Conditions unfolded diagram in OPM

opposed to informatical ones) are shaded. In order to keep the SD clear and focus only on top-level entities, it does not contain the components of the Driving Conditions object. A separate diagram, shown in Figure 4, was created for that purpose, using OPM's unfolding mechanism.

We continue to construct the model by specifying use cases. The SysML model in (OMG 2007c) includes two use case diagrams. The first one, showing top-level use cases, is not brought in this paper. The second diagram, shown in Figure 5, depicts goal level use cases associated with "Operate the Vehicle", which is one of the top-level use cases. While in SysML the use cases are modeled by a dedicated diagram type, in OPM the semantically equivalent information is expressed using the single OPD diagram kind. The diagram describing the top-level use cases, which is shown in Figure 6, was created in OPM by zooming into the Vehicle Using process from the top-level SD, exposing its subprocesses. These subprocesses are the top-level system use cases. The Hybrid SUV object is connected with an instrument link to the containing Vehicle Using process, as it serves as an enabler to all the subprocesses it contains. Similarly, further decomposition was done by zooming into the Vehicle Operating process, resulting in the OPD of Figure 7. In OPM, subprocesses within a containing process are executed by default from top to bottom, so since Driving appears above the Parking the former happens first. Obviously, we used here information we know from our everyday life, as this information is not part of the original SysML use case diagram. Further refinement of the Driving process, which

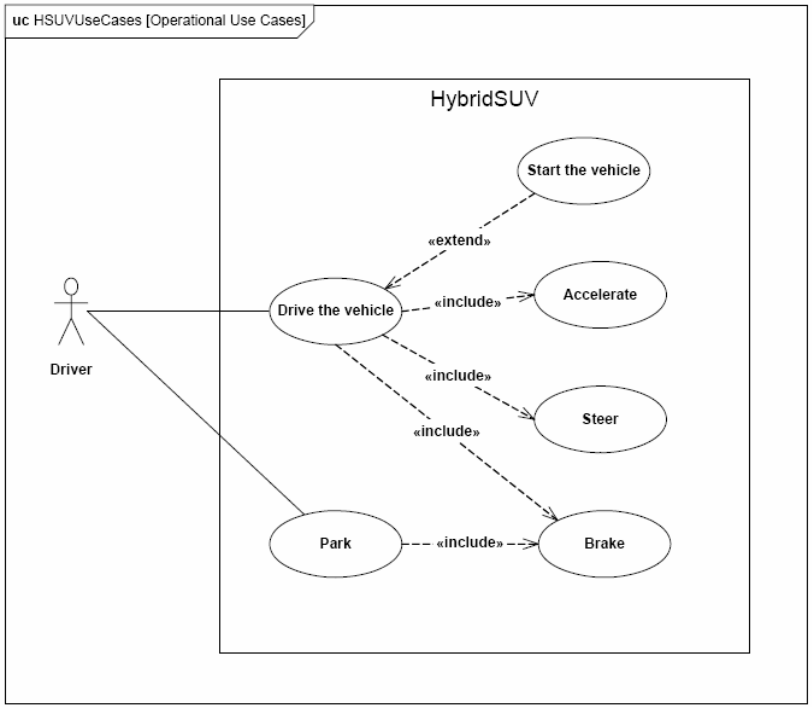

Figure 5. Operational use cases diagram associated with "Operate the Vehicle" in SysML

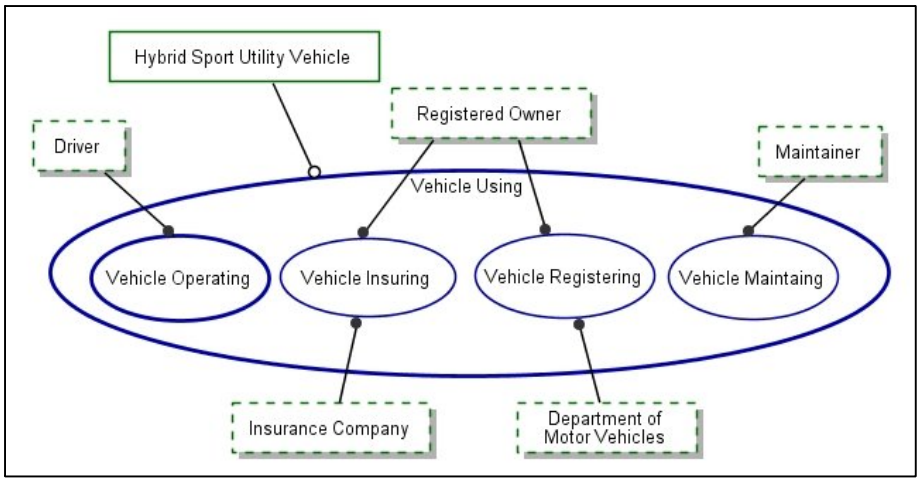

Figure 6. The Vehicle Using process in-zoomed in OPM

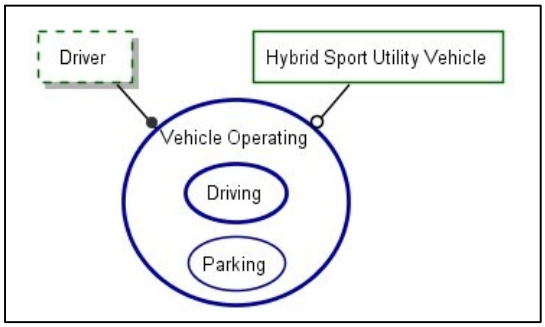

Figure 7. The Vehicle Operating process in-zoomed in OPM

depicts the rest of the information contained in the operational use cases SyML diagram, is brought in the next subsection, as it contains additional information.

System Context, Boundaries and Use Cases Summary: In this subsection we presented part of the system model describing the system context and the main use cases in both SysML and OPM. Although we captured only a small part of the model, we can already make several observations. Even though we have been concerned with just one aspect of the system, the SysML description already required using two types of diagrams—internal block diagram, used as the context diagram, and use case diagram. Each type has its own graphical format and set of symbols. In contrast, the same information was expressed in OPM using the single OPM diagram type, OPD.

As OPM defines explicit and consistent abstraction-refinement mechanisms to manage complexity, the model is presented gradually with a clear hierarchy of OPDs. Specifically, we saw the top-level diagram (SD) and two of its descendant OPDs: Driving Conditions unfolded and Vehicle Using in-zoomed. In SysML, although hierarchy among diagrams does exist, e.g., top-level use cases and goal-level use cases, there is no built-in structured mechanism to manage the hierarchy as it exists in OPM. With OPM the whole system model is the OPD set, which is organized as a directed acyclic graph, so it is always clear how diagrams are related to each other, and what is the detail level of each diagram in the "big picture". Although these observations were derived from investigating the system context and use cases only, they are general and not limited to just this aspect of the model. As demonstrated in subsequent subsections, these are typical characteristics of these two languages.

System Behavior – Elaboration

After specifying the system context and its main use cases, we further model and analyze the top-level system behavior. Figure 8 shows the OPD in which the Driving process was zoomed into. It contains the remaining information from the SysML operational use cases diagram of Figure 5 plus part of the state information described in a separate SysML state machine diagram, not included in this paper. As the condition link in the OPD shows, the Vehicle Starting subprocess occurs if the operational status of the vehicle is off. As a result of this subprocess occurrence, the vehicle's operational status is changed to operate. This diagram demonstrates for

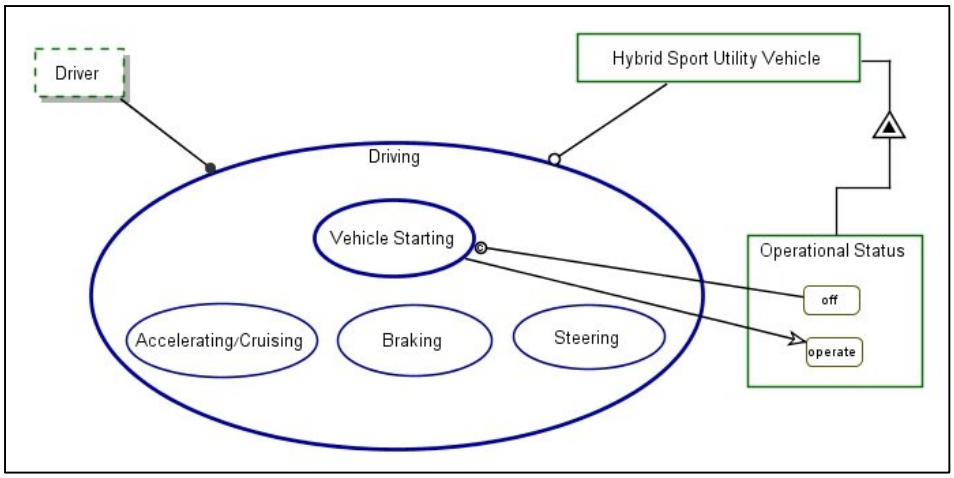

Figure 8. The Driving process in-zoomed in OPM

the first time the combination of objects, processes, and states in a single diagram.

To further model the behavior of "start the vehicle" use case, the reference SysML model introduces two sequence diagrams: (1) "start the vehicle" black-box diagram (not shown), in which the lifelines are at the whole system level, and (2) "start the vehicle" white-box diagram (Figure 9), which decomposes the lifelines of the Hybrid SUV system into components.

In order to equivalently refine the model in OPM, we again zoom into the Vehicle Starting process, yielding a new OPD, shown in Figure 10. The resulting OPD combines the details modeled in the "black-box" and "white-box" SysML sequence diagrams, as well as part of those modeled in an additional SysML state machine diagram. The OPD also depicts the aggregation hierarchy of the relevant objects (including the "mid-level" Power Subsystem object), which is missing in the aforementioned SysML diagrams. Obviously, the OPM diagram looks pretty different from the corresponding SysML diagrams in this case, due to dissimilar graphical layout and notation. There are other cases, demonstrated in the next subsection, in which diagrams of these two languages look much more similar. In general, structural OPM and SysML diagrams will look similar, because they show relation among entities (blocks, objects), while procedural (behavioral) OPM and SysML diagrams will look dissimilar, since unlike in OPM, there is no single SysML diagram type that shows all the dynamic aspects of the system.

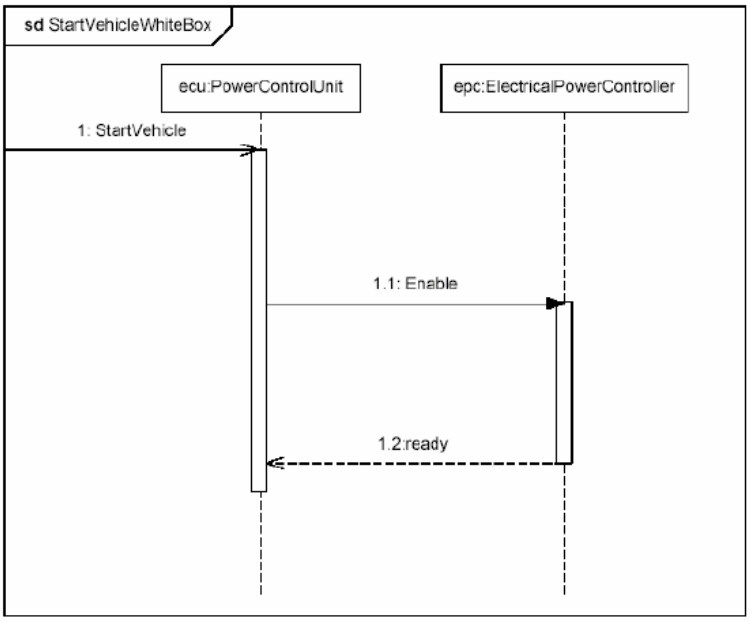

Figure 9. While-box “StartVehicle” sequence diagram in SysML

As noted above, OPM supports textual representation of the model complementary to the graphical description. The text, described in OPL, is matched with each of the OPDs in the model. The OPL text, generated automatically by OPCAT, for the OPD in Figure 10 is listed in Figure 11. Corresponding OPL paragraphs are generated for each and every OPD in the model.

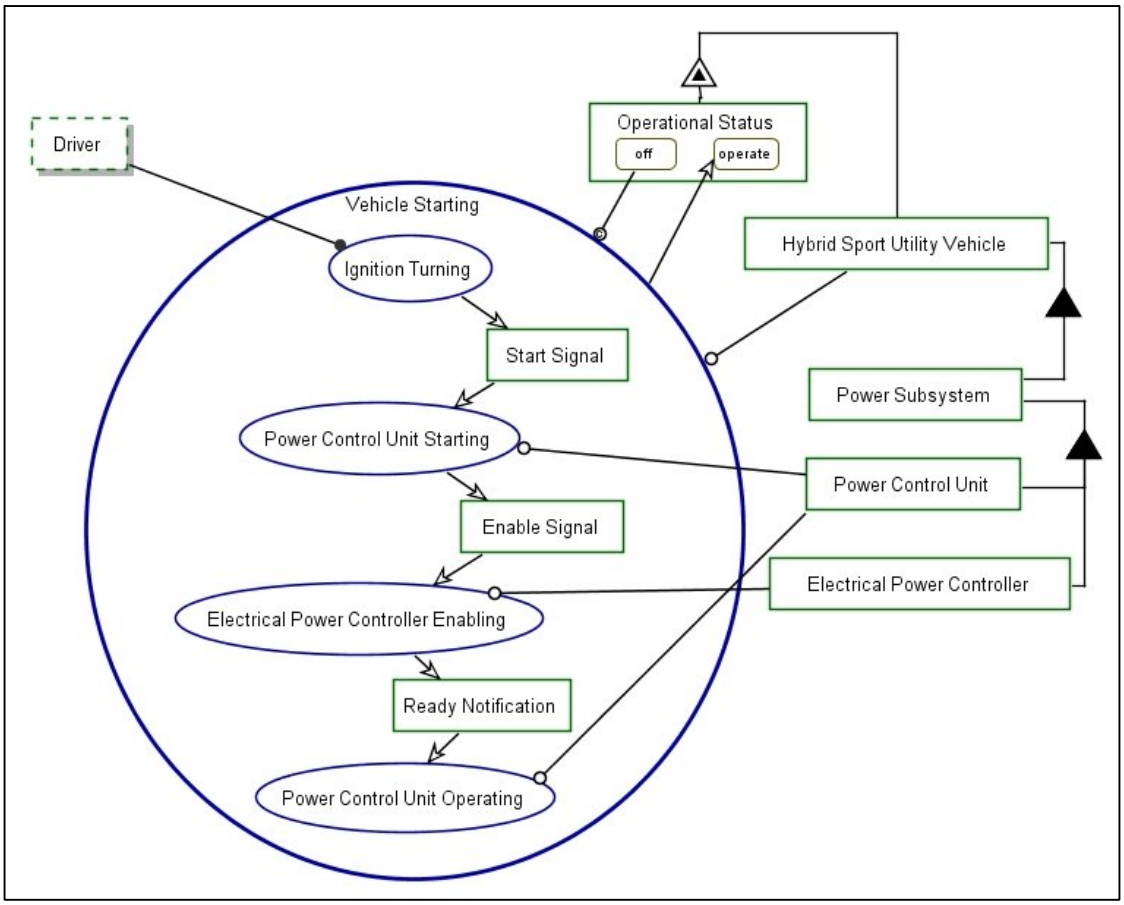

Figure 10. The Vehicle Starting process in-zoomed in OPM

System Behavior Elaboration Summary: In this subsection we focused on system behavior by elaborating on the previously modeled use cases. The OPM refinement-abstraction mechanism (specifically in-zooming) has been employed again several times, resulting in a structured system map. The portion of the system model presented in this subsection further demonstrates the benefits of this OPM feature, allowing easy navigation and consistency. As the model becomes more complex, this advantage becomes more noticeable.

Another characteristic mentioned earlier but illustrated more intensely here is OPM's single diagram type approach compared with the model multiplicity method of SysML. Objects, processes, and states are modeled together in OPM using the unified Object-Process Diagram, as opposed to multiple types of diagrams required for the same task in SysML. The OPM approach has several advantages, primarily the ability to model real world systems, where structure and behavior need to be shown concurrently. The use of just one type of diagram also makes OPM simpler and easier to learn and use. However, languages with multiple diagram types like SysML allow using dedicated diagram types for different aspects of the model, possibly making it easier to understand diagrams of specific aspects in certain cases.

Both SysML and OPM employ graphical diagrams for model representation. However, OPM features also complementary text-based description of the model in human (and machine) readable form, namely OPL. As indicated in Grobshtein et al. (2007), the dual visual-textual modalities representation aids human comprehension of the model. Moreover, the textual

representation can be helpful in cases where some of the system development stakeholders, such as domain experts and non-technical customers, are not familiar with the graphical modeling language. Still they are supposed to review and evaluating the model, and approve it or comment on it. Since the OPL paragraphs can be generated automatically from the model diagrams by software tools (like OPCAT), the visual and textual modalities are always synchronized with no additional efforts or overhead.

Hybrid Sport Utility Vehicle exhibits Operational Status.
Operational Status can be off or operate.
Hybrid Sport Utility Vehicle consists of Power Subsystem.
Power Subsystem consists of Power Control Unit and Electrical Power Controller.
Driver is environmental and physical.
Driver handles Ignition Turning.
Vehicle Starting occurs if Operational Status is off.
Vehicle Starting requires Hybrid Sport Utility Vehicle.
Vehicle Starting yields operate Operational Status.
Vehicle Starting zooms into Ignition Turning, Power Control Unit Starting, Electrical Power Controller Enabling, and Power Control Unit Operating, as well as Ready Notification, Enable Signal, and Start Signal.
Ignition Turning yields Start Signal.
Power Control Unit Starting requires Power Control Unit.
Power Control Unit Starting consumes Start Signal.
Power Control Unit Starting yields Enable Signal.
Electrical Power Controller Enabling requires Electrical Power Controller.
Electrical Power Controller Enabling consumes Enable Signal.
Electrical Power Controller Enabling yields Ready Notification.
Power Control Unit Operating requires Power Control Unit.
Power Control Unit Operating consumes Ready Notification.

Figure 11. The OPL paragraph that describes textually the OPD in Figure 10

System Structure – Hierarchy and Interconnection

SysML defines two primary diagram types for structure specification. The first one is Block Definition Diagram (BDD), used to describe relationships among blocks, such as hierarchies. The second diagram type is Internal Block Diagram (IBD), which describes the internal structure of a block, usually through parts and connectors. Figure 12 contain the major components of the

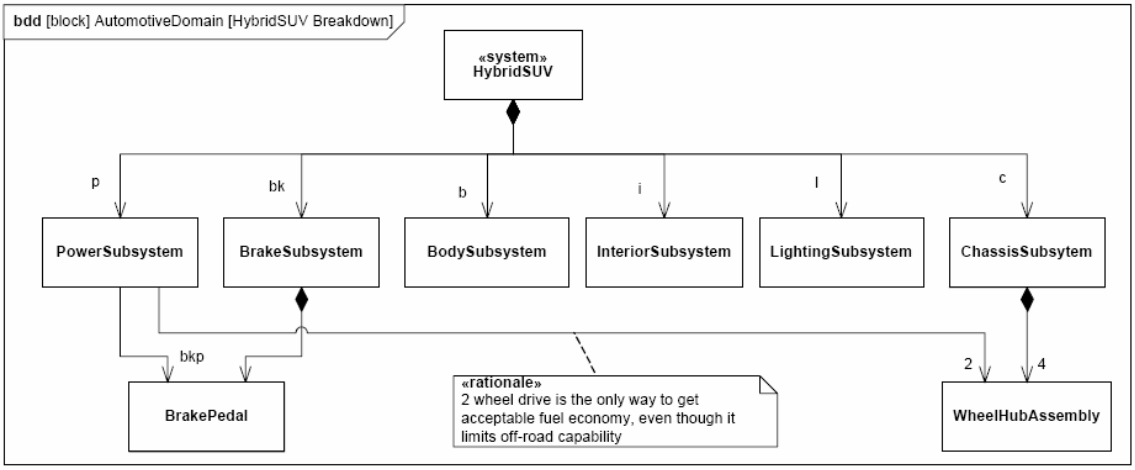

Figure 12. Structure of the Hybrid SUV system in SysML

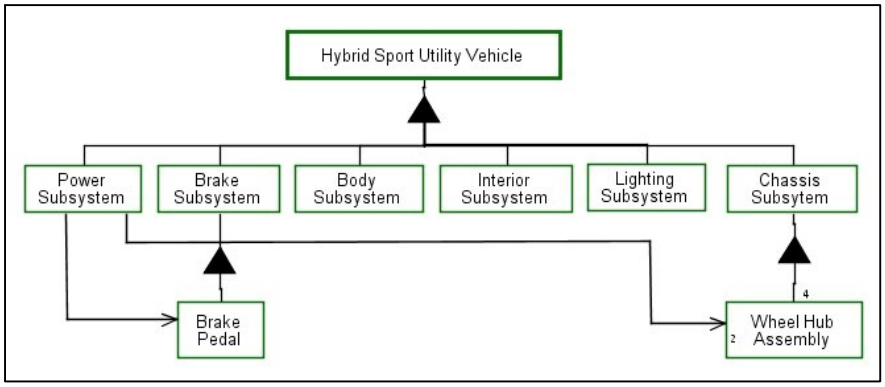

Figure 13. Structure of the Hybrid SUV system in OPM

Hybrid SUV system in SysML as a Block Definition Diagram, while Figure 13 is the corresponding OPD. These two diagrams are very similar, capturing the same kind of information.

Figure 14 shows how the top-level components of the system are connected together as a SysML Internal Block Diagram. The corresponding OPD, shown in Figure 15, uses the bidirectional structural relation for modeling the relationships between the objects, annotated with the "connected" tag. Here too, the corresponding SysML and OPM diagrams look similar, and they are indeed structural as noted above.

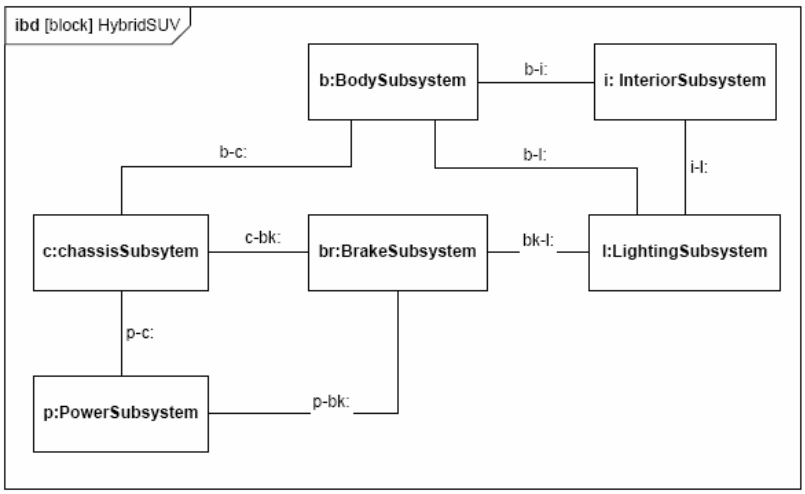

Figure 14. Internal structure of the Hybrid SUV in SysML

System Structure – Hierarchy and Interconnection Summary: Description of the Hybrid SUV major structural constructs, namely hierarchies and interconnections, yields very similar diagrams in SysML and OPM. While SysML employs two different diagram kinds, BDD and IBD, in OPM this can still be expressed with the same single diagram type, OPD, using its compact set of graphical notations. Due to its build-in abstraction-refinement mechanism, OPM's diagram hierarchy is also beneficial in the context of system structure. For example, the OPD of Figure 13 is a descendant of SD, the top-level OPD, since if refines SD by unfolding the Hybrid SUV object. The same OPD is an ancestor of another OPD (not provided in this paper), in which the Power Subsystem object is unfolded to expose its

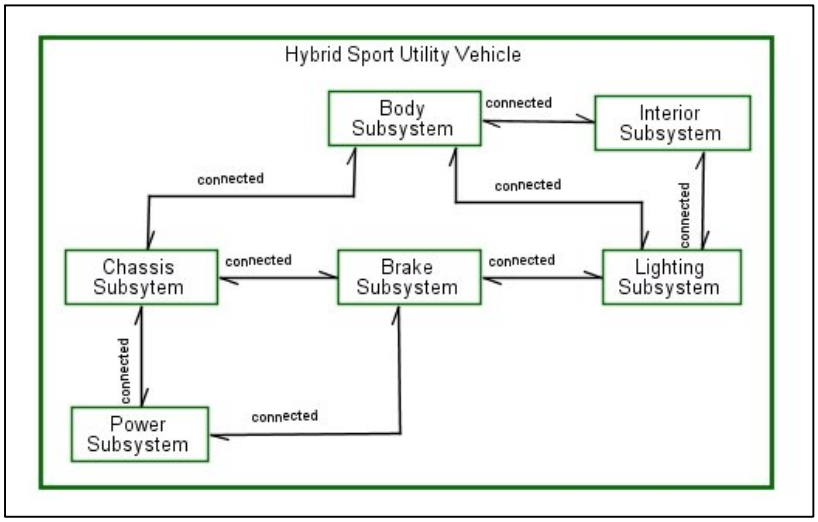

Figure 15. Internal structure of the Hybrid SUV in OPM

parts. One can drill down the hierarchy as much as needed, until the finest required level of detail is reached.

Case Study Summary and Discussion

Through partial modeling of the Hybrid SUV sample system we have demonstrated and analyzed similarities and differences between SysML and OPM, two conceptual systems modeling languages. While in some language parts, such as the structural decomposition, we found the languages to be similar, in other areas, notably behavior, differences were encountered.

Maybe the most recognizable differentiator between these languages is SysML's model multiplicity—the fact that it consists of nine types of diagrams, compared with OPM's single diagram type. As illustrated, this factor has various implications on the resulting system model. The approach of OPM allows holistic systems modeling with simple and consistent means to combine structure and behavior naturally. SysML, on the other hand, advocates the use of different diagram types for modeling different aspects of the system. As each aspect is modeled using a different diagram, SysML can sometimes provide visual representation that is more focused on the objective at hand. For instance, sequence diagrams for object interactions provide a clear description of the sequence of events and messages exchanged among the participating objects. Although dedicated diagrams for certain aspects can be helpful, dealing with a relatively large number of diagram types, each with its dedicated set of symbols, renders the language complicated. Consequently, the effort to learn the language and understand the diagrams depicted in that language increases.

The importance of modeling hierarchies within a system model has been demonstrated in both structural and behavioral contexts. The abstraction-refinement mechanisms built into OPM provide a consistent framework for managing complexity in a hierarchical manner. While simple and intuitive, they provide a powerful way to organize the model and navigate through its detail levels. SysML also provides for hierarchical modeling, but it is not as clear and simple as it is in OPM. The existence of multiple types of diagrams inevitably adds inherent complexity. Appropriate support by software tools may be helpful in this area.

We focused on the three fundamental system model aspects, namely the functional, behavioral and structural ones. Model-related requirements management and engineering analysis (e.g., performance) are additional aspects, which were given much attention in SysML. In accordance with the model multiplicity approach, it is not surprising that SysML support in these aspects include dedicated diagram types, namely Requirement Diagram and Parametric Diagram. The engineering analysis modeling capabilities supported by SysML through parametrics and constraint blocks have no direct equivalent constructs in OPM. It is possible to model mathematical expressions in OPM using "regular" objects and processes, as explained and demonstrated by Dori (2002). Nevertheless, SysML mechanisms aimed specifically at this aspect are usually more suitable and convenient to use. Hence, SysML is currently more attractive in modeling systems where significant engineering analysis is required.

SysML supports requirements management by the Requirements Diagram, as well as by combining requirements artifacts and other model elements using (possibly other) diagrams. In SysML, the connections between different system components and the requirements satisfied by them are visualized. OPM does not explicitly include the notion of requirements; however OPCAT, OPM-supporting modeling tool, has facilities for requirements traceability and management. In contrast to SysML, in OPCAT requirements traceability is done by linking text to the graphics. The modeler monitors connections between OPM design artifacts and the

matching requirements through requirements coverage tables that are built into OPCAT. Unlike the SysML requirements/components diagrams, OPM deals with the issue by separating traceability information from the graphical representation. It is also possible to treat requirements as any other informatical objects in the system. Using this method, modeling semantics such as requirements hierarchies, dependencies and interrelations with other model elements can be easily performed with the usual OPM building blocks.

Conclusions and Future Work

The modeling of the sample Hybrid SUV system and the resulting analysis and discussion show that neither of the examined languages is by all means better than the other. Both SysML and OPM have relative strengths and weaknesses, so choosing the appropriate modeling language depends on the problem at hand. In OPM, the well-organized hierarchy of the single diagram type, OPD, promotes smooth holistic understanding of the system and its environment, with concurrent representation of both the structure and the behavior of the system. However, with OPM it might be more difficult to express certain fine points of a complex system, such as those permitted by the large variety of SysML diagrams. Specifically, the requirements and parameterics support allow easier modeling of these aspects in SysML. Conversely, a SysML model, spread over a number of different diagram types tends to be complicated and requires considerable time and effort to both create and understand.

Considering the benefits and drawbacks of each of the two languages compared, ways to create synergies between them should be explored. One possible approach is to extend SysML with some key OPM features. Important OPM features include text description of the model (OPL) and a structured hierarchical modeling mechanism. Adding a new diagram type to SysML combining structure and behavior, based on the Object-Process Diagram, will enable a more holistic understanding of the system's structure and behavior. Furthermore, existence of such diagram in SysML can lead to simplification of the language, as it will provide for eliminating several diagram types and graphical symbols that will become redundant. These issues should be among those considered during the development of the next SysML versions.

Analogically, OPM can also benefit from adopting valuable missing features from SysML. A significant aspect to be handled is the modeling engineering analysis, which can be addressed by extending OPM capabilities to make it executable. Additional development will be needed to incorporate other features supported by SysML, such as certain properties of the requirements modeling and missing "fine points" constructs. Moreover, to enable standard data exchange and collaboration with SysML-based tools, automatic generation of equivalent SysML model in XMI format (OMG 2005) from within the OPM-supporting tool should be investigated.

References

Bock, C. 2006. SysML and UML 2 Support for Activity Modeling. *Journal of International Council of Systems Engineering*, vol. 9 no. 2, pp. 160-187.

Dori, D. 2002. *Object-Process Methodology – A Holistic Systems Paradigm*. Springer-Verlag.

Dori, D., Reinhartz-Berger, I., and Sturm, A. 2003. Developing Complex Systems with Object-Process Methodology using OPCAT. Conceptual Modeling – ER 2003, Lecture Notes in Computer Science (2813), pp. 570-572. http://www.opcat.com.

Friedenthal, S., Moore, A., and Steiner, R. 2007. OMG Systems Modeling Language. Tutorial presented at the Seventeenth Annual International Symposium of the International Council on Systems Engineering (INCOSE), June 24-28, in San Diego, CA. Available at

http://omgsysml.org/INCOSE-2007-OMG-SysML-Tutorial.pdf.
Grobshtein, Y., Perelman, V., Safra, E., and Dori, D. 2007. Systems Modeling Languages: OPM Versus SysML. In Proceedings of the International Conference on Systems Engineering and Modeling, March 20-23, in Herzeliya and Haifa, Israel.
Mayer, R. E. 2001. *Multimedia Learning*. Cambridge University Press, New York.
OMG – Object Management Group. 2003. UML for Systems Engineering RFP (OMG document number ad/2003-03-41). Available at http://www.omg.org/cgi-bin/apps/doc?ad/2003-03-41.pdf.
———. 2005. MOF 2.0/XMI Mapping Specification v2.1 (OMG document number formal/05-09-01). Available at http://www.omg.org/cgi-bin/apps/doc?formal/05-09-01.pdf.
———. 2007. Unified Modeling Language: Superstructure version 2.1.1 (OMG document number formal/07-02-05). Available at http://www.omg.org/cgi-bin/apps/doc?formal/07-02-05.pdf.
———. 2007. Unified Modeling Language: Infrastructure version 2.1.1 (OMG document number formal/07-02-06). Available at http://www.omg.org/cgi-bin/apps/doc?formal/07-02-06.pdf.
———. 2007. OMG Systems Modeling Language Specification V1.0 (OMG document number formal/2007-09-01). Available at http://www.omg.org/cgi-bin/apps/doc?formal/07-09-01.pdf.
———. 2007. OMG Systems Modeling Language website. http://omgsysml.org.
Peleg, M., and Dori, D. 2000. The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. *IEEE Transactions on Software Engineering*, 26, 8, pp. 742-759.

Biography

Yariv Grobshtein is an M.Sc student in the Information Systems Engineering Area at the Technion – Israel Institute of Technology. He holds a B.Sc in Computer Science and an MBA from the Technion, and has 10 years of professional experience in software rich systems development. His M.Sc research subject is in the area of systems modeling languages.

Dov Dori is Associate Professor and Head of the Information Systems Engineering Area at the Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, and Research Affiliate at Massachusetts Institute of Technology. Prof. Dori has developed Object-Process Methodology (OPM), a holistic systems paradigm for conceptual modeling, presented in his 2002 book (by Springer). Prof. Dori is Fellow of the International Association for Pattern Recognition and Senior Member of IEEE and ACM. He authored six books and over 100 journal publications and book chapters.