Systems Modeling Languages: OPM Versus SysML

Yariv Grobshtein, Valeriya Perelman+, Eliyahu Safra, Dov Dori+
Technion, Israel institution of Technology, Haifa, Israel
{yarivg, valeriya, safra, dori}@technion.ac.il

Abstract

As systems are becoming ever larger and more complex, and as more stakeholders, typically from different disciplines, are involved throughout the system lifecycle, the challenge of overcoming the complexity inherent in systems development grows too. While a document-centric approach has been common practice, coping with the growing complexity of current systems calls for a model-based approach.

This work discusses two systems modeling languages, OMG's SysML – Systems Modeling Language and OPM – Object Process Methodology. To demonstrate the similarities and differences between the languages, a concrete sample system was modeled in both SysML and OPM and the issue was discussed in an experimental graduate students' web-based forum as part of a course in Information Systems Engineering. Our study shows that SysML tends to be more appropriate in cases where a detailed picture is required. Alternatively, OPM is more suitable for defining system boundaries and demonstrating the overall picture of the system.

1. Introduction

One of the main goals of conceptual modeling is to improve analysis and enable detailed, deep design. Early model construction provides for verification and validation of a system before its implementation and integration. 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.

The Unified Modeling Language (UML) [1], [2] is widely used in the software engineering discipline. However, using the same language for system modeling may be problematic due to several reasons. Since UML's focus is on software, its ontology and taxonomy are limited to software artifacts. For example, physical characteristics and components of a system are hardly expressed in UML diagrams. UML does not adequately support modeling of hierarchies within a system model, an issue which is typically of greater importance in systems engineering than in software.

Several modeling languages have been designed for general-purpose systems, each with its semantics and notation. Typically, a System Modeling Language (SML) provides tools for representing a system in both textual and visual modalities. A key to successful model construction is choosing the appropriate SML.

This work discusses two of the state-of-the-art representative SMLs: OMG Systems Modeling Language (SysML) [3], [4], [5] and Object Process Methodology (OPM) [6]. SysML is based on UML, 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. Both system types can comprise physical and informatical things (objects and processes).

SysML includes several types of UML diagrams as well as new types of diagrams for systems engineering, such as Requirement Diagram and Parametric Diagram. In contrast, OPM uses a single type of diagram, in which some elements that are important for systems engineering (such as the SysML parametric constraints) are missing. Both languages support hierarchical representation of the model. However, in contrast to SysML, where the model is represented in separate

+ This author was partially supported by the Bernard M. Gordon Center for Systems Engineering.

views with partial support of hierarchy, in OPM the entire system model is built in one well-defined hierarchy. These are but few of several dissimilarities between the languages, which make it interesting to study and compare them, and to define guidelines regarding the selection of the more appropriate modeling environment in different cases.

To investigate the similarities and differences between the languages, a concrete system example has been modeled in both SysML and OPM. The example selected to be modeled is a distiller system. The specification and the SysML model of the distiller system was taken from [5], and an OPM model was created for the same system specification. While in [5] a top-down model construction approach was taken using SysML, we show a typical middle-out model construction in OPM. To probe further into the research subject, a students' Web-based forum on the topic "Systems Modeling Languages" was conducted as part of the graduate course “Methodologies for Information Systems Development” at the Technion during the Spring 2006 semester. This hybrid course along with students' feedback is described in detail in [7]. One of the discussion subjects, comparing SysML and OPM, is summarized in this paper.

2. Overview of OPM and SysML

As a prelude to our comparative investigation, we briefly describe the two languages considered in this work: OPM and SysML. Note that the SysML description is longer, as the language employs larger number of diagrams.

2.1. Object Process Methodology

Object Process Methodology (OPM) is a holistic approach for 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 textually 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.

The ontology comprises three different types of entities: objects, processes (both referred to as "things") and states. Objects exist for some times in the system, and if they are stateful (i.e., have states), then they are at some state at any point in time. Processes transform objects: they generate and consume objects, or affect stateful objects by changing their state. Both objects and processes can be involved in the four OPM fundamental structural relations – generalization, aggregation, instantiation, and exhibition. Objects can also be structurally related to each other by tagged relations by either unidirectional or bi-directional relations, similar to association links in UML class diagrams. The OPM ontology also includes several types of procedural links between processes and objects or their states, including effect, consumption, agent, instrument, and result links, as well as invocation and time exception links between processes.

An OPM model consists of hierarchically organized OPDs. 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 refinable (part, specialization, feature, or instance) of a thing at a higher abstraction level (a more abstract OPD). It is sufficient for some detail to appear once in some OPD for it to be true for the system in general even though it is not shown in any other OPD. 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.

OPCAT [8] is a CASE tool that supports OPM-based system development and lifecycle management. This software package implements the 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. A detailed example of an OPM model developed in OPCAT, including its SD and one level of in-zooming is shown in Section 3.3. The example illustrates both objects and processes and the interrelations among them. The model is expressed both graphically as a set of OPDs and textually, as a collection of OPL sentences.

2.2. SysML

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 Request for Proposal (UML for SE RFP) [9] 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. Three years after the RFP was published, version 1.0 of the SysML specification was adopted by OMG in May 2006 [3].

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. [5] contains examples of some SysML key diagram types.

Overall, SysML includes nine types of diagrams: four types of structure diagrams, four types of behavior diagrams, and a requirement diagram. In this section, we discuss mainly the extensions and modifications of SysML compared with UML which are relevant to 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 with no change.

2.2.1. 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. A Requirement Diagram example is presented in Figure 2 in its graphical format. 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 requirements among relationships, such as requirements hierarchies, source-derived requirement dependencies, satisfaction relations between requirements and the model, and verification dependencies to test-cases.

2.2.2. 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. 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.

2.2.3. Parametric. Parametric Diagram is the second new diagram type 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 mass of a vehicle).

2.2.4. Behavior. SysML specifies four types of behavioral diagrams: Activity Diagram, Sequence Diagram, State Machine Diagram and Use Case Diagram. An activity is the fundamental behavioral element which is used in the various SysML behavioral diagrams (excluding the 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, 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.

3. The Distiller Example

This section contains a real-world example. The example demonstrates some of the differences between SysML and OPM. We use a relatively small example in order to show different model aspects in a united and compact way. The model description is accompanied with a typical model construction process of each SML.

3.1. The Distiller System

Distiller is a system for purifying dirty water. The Distiller consists of three main components: (1) Counter Flow Heat Exchanger – for heating dirty water and condensing steam, (2) Boiler – for boiling the dirty water, and (3) Drain – for draining residue.

The purifying process starts with heating the dirty water in the Counter Flow Heat Exchanger. Then, the Distiller boils the dirty water in the Boiler. Finally, residue is drained by the Drain. The first heating stage uses steam that was created in previous stages, and condenses the steam to water, so the condensed steam is the output of the system. Figure 1 shows a basic behavior diagram of the distiller. The rectangles stand for the system processes, whereas the ellipses stand for process inputs and outputs.

3.2. Modeling the Distiller with SysML

In this section we describe how the distiller is modeled in SysML. Presenting some of the diagram types, we describe a typical modeling process. A complete model and a detailed description of the modeling process can be found in [5]. SysML modeling is a three-step process. The first step includes identifying and organizing the needed libraries, resulting in a list of requirements along with assumptions. In the second step, the behavior and structure of the model are defined and a constraint list is compiled. Finally, in the third step, the model is checked against the requirements and constraints. The second and third stages are repeated until the model conforms to all the requirements and constraints. Following is a description of each of the three steps of the distiller system modeling.

In the first stage, a Package Diagram and a Requirement Diagram are built. The Package Diagram contains six groups: Distiller requirements, Distiller use case, Distiller structure, Distiller behavior, Value types, and Item types. The Requirement Diagram contains two parts: one for the distiller specifications and one for assumptions. The requirements diagram is shown in Figure 2.

In the second stage, the structure and behavior of the system are modeled. This is done using two diagrams: Activity Diagram (behavior) and Block Definition Diagram (structure). The distiller behavior shown in Figure 3 is a formalization of Figure 1. The richer symbol set of SysML enables clear definition of the start and end states, as well as clear differentiation between sequence connectors (the dashed arrows) and input/output flows (the straight lines).

The distiller block diagram contains three main blocks: Heat Exchanger, Boiler, and Valve. It is

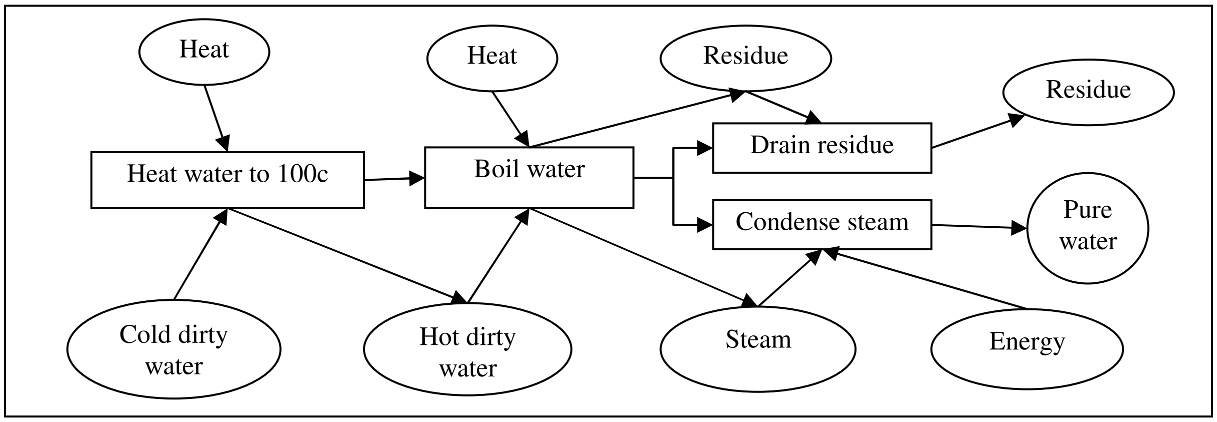

Figure 1: Distiller basic behavior diagram

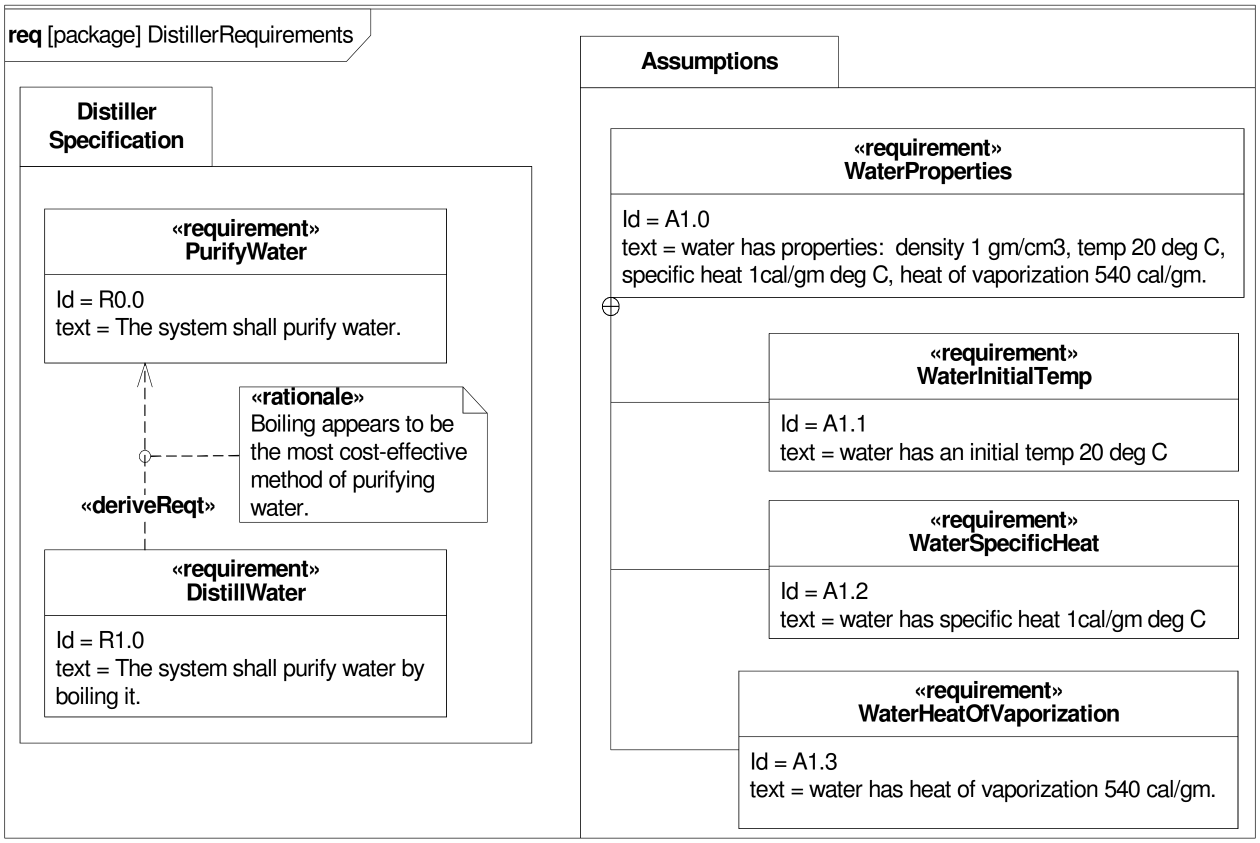
req [package] DistillerRequirements
Distiller Specification
«requirement» PurifyWater
Id = R0.0
text = The system shall purify water.
«rationale» Boiling appears to be the most cost-effective method of purifying water.
«deriveReqt»
«requirement» DistillWater
Id = R1.0
text = The system shall purify water by boiling it.
Assumptions
«requirement» WaterProperties
Id = A1.0
text = water has properties: density 1 gm/cm3, temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.
«requirement» WaterInitialTemp
Id = A1.1
text = water has an initial temp 20 deg C
«requirement» WaterSpecificHeat
Id = A1.2
text = water has specific heat 1cal/gm deg C
«requirement» WaterHeatOfVaporization
Id = A1.3
text = water has heat of vaporization 540 cal/gm.

Figure 2: Distiller Requirement diagram in SysML

possible to connect between the block diagram and the activity diagram by allocations, which are used to allocate behavior onto structure and flow onto I/O. Other diagrams such as sequence diagram and state-machine diagram may be built in order to complete the model. The second stage ends when a complete model is obtained.

In the third stage the model is tested against the requirements and assumptions of the system. The existence of the visual model helps understand the relations and dependencies between the system processes. When a modification is required, the system

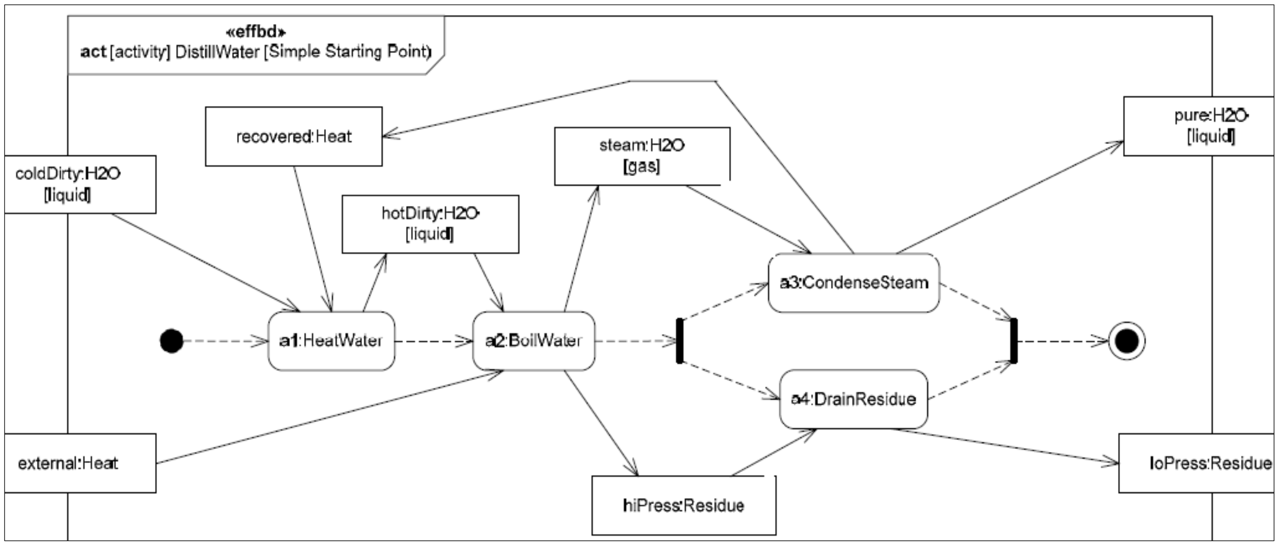

Figure 3: Distiller Simple Behavior diagram in SysML

modeler should return to the second stage and changes the structure and the behavior as needed. Stages two and three are done repeatedly until the system satisfies all the requirements and constraints.

3.3. Modeling the Distiller with OPM

In this section we briefly delineate an OPM-based distiller system modeling process, which is typical of OPM modeling, and present the resulting model. Like the modelers of the SysML distiller model, the process starts with the distiller requirements definition. This stage was completed by inserting a well-organized hierarchical tree of requirements into OPCAT requirements module. The module assigns a unique key for each requirement and enables traceability of the requirements to the designed model. Throughout the model development, each newly introduced design artifact was linked to all the related requirements.

The next stage included definition of the system boundaries: defining its main process, which is the major function of the system being modeled—in our case Purifying. Around this process the involved objects are depicted: affected object (Water in our case), inputs and outputs, main agents (OPM actors) if they exist, and possible environmental (external) objects and processes involved. Figure 4 depicts the System Diagram (SD, the root of the OPD hierarchy tree) of the distiller system. This top-level diagram emphasizes Purifying as the system's function, its major goal. This is typical of the top-level OPD of any OPM model: The sole process it displays is the system's function.

Next, hierarchical refinement of the system processes and objects takes place concurrently, using the refinement mechanisms of in-zooming for processes and unfolding for objects. Figure 5 shows the Purifying process in-zoomed. A detailed sequence of subprocesses included in the purifying process, shown inside the Purifying ellipse, includes Heating, Boiling, Condensing, and Draining. The Y-axis within an in-zoomed process is its timeline—it specifies the sequencing of the operations: Heating, placed at the top is the first operation. It is followed by Boiling, Condensing, and finally Draining. The diagram also specifies the structure of the distiller by showing its parts: Heat Exchanger, Boiler, and Valve. Each subprocess uses a different subset of parts of the Distiller and changes the state of Water from dirty to hot, boiling, and finally pure.

The 3D shading effect of the entities in the OPDs symbolize that they are physical. The dashed contour of the Water and External Heat objects denotes that they are environmental, i.e. not part of the system. This distinction clearly delineates the border between the modeled system and the external entities, which are part of the system's environment. OPM supports textual

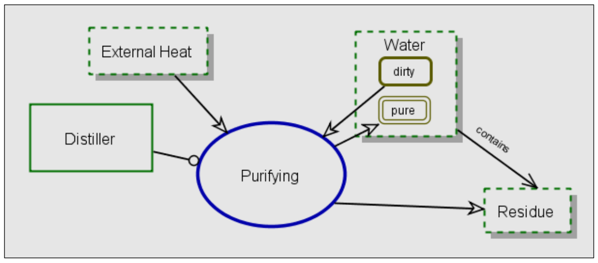

Figure 4: The System Diagram of the Distiller System in OPM

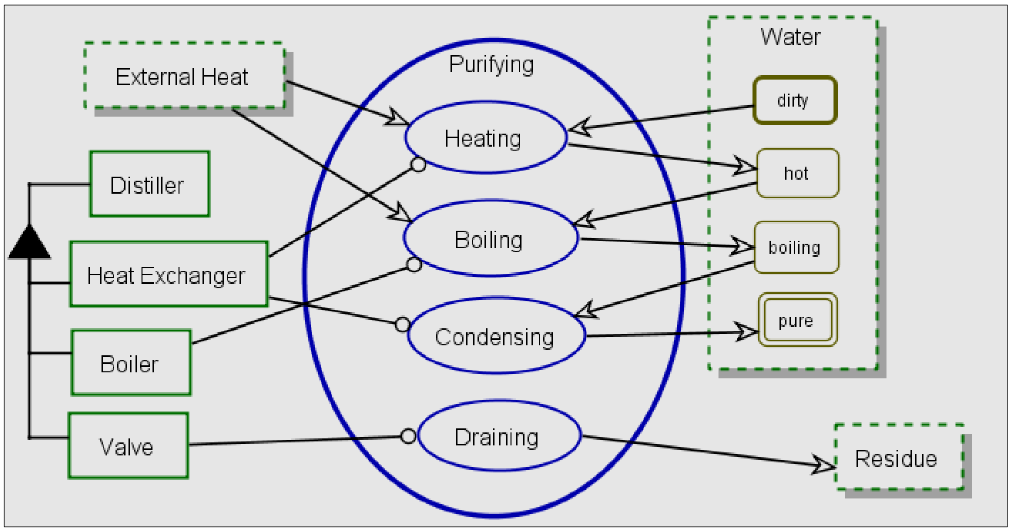

Figure 5: In-zoomed Purifying process in OPM

representation of the model additionally to the graphical description. The OPL text for the OPD in Figure 5 is listed in Figure 6.

External Heat is environmental and physical.
Water is environmental and physical.
Water can be dirty, pure, boiling, or hot.
    dirty is initial.
    pure is final.
Residue is environmental and physical.
Distiller consists of Heat Exchanger, Boiler, and Valve.
Purifying changes Water from dirty to pure.
Purifying yields Residue.
Purifying zooms into Heating, Boiling, Condensing, and Draining.
    Heating requires Heat Exchanger.
    Heating changes Water from dirty to hot.
    Heating consumes External Heat.
    Boiling requires Boiler.
    Boiling changes Water from hot to boiling.
    Boiling consumes External Heat.

Figure 6: The OPL paragraph that describes textually the OPD in Figure 5

4. Forum Summary

As part of the study, a forum on systems modeling languages was conducted within the graduate course titled "Methodologies for Information Systems Development" of the Information Systems Engineering track at the Technion, Israel Institute of Technology. Prerequisites of the course, in which 19 students participated, included Systems Analysis and Software Engineering courses, where the students study and practice UML and OPM. To complete the background of the students, they were asked to read the OMG SysML tutorial [5] and visit the official OMG SysML website [4]. Students were verbally encouraged to express any thoughts they have on the given topics.

The forum summary reveals interesting user judgments on systems modeling languages. The following subsections summarize the original forum subjects. Different interesting ideas in each topic were selected and abridged.

4.1. Graphical Versus Textual Information Representations

Most modeling languages comprise both textual and graphical representation in various proportions. The argument focused on implications of using each one of these two modalities (representation modes) and on which approach is preferable for what purposes. Table 1 summarizes benefits of each modality as agreed by the majority of the forum participants.

Table 1: Textual vs. graphical representations summary

Textual Representation	Graphical Representation
Expression of many details in a relatively small space	An easy way to get a general view of the system
Depicting constrains that are hardly expressible in the graphical representation	Easy to depict different relations among system components
Very flexible and can also include some formalism (mathematics, etc.)	The representation is usually more structured and formal
Must be read in a predefined order	Can be interpreted in a "random access" mode

As Table 1 shows, students expect system details, such as different constrains, to be specified in text, whereas the general system view is better represented visually. The selected type of representation also depends on the desired level of detail, so the target audience plays an important role in the modality selection. Technical people may need more detailed system description, so they should be provided with more written information. However, non-technical people, who can always read text, often find it difficult to comprehend diagrams. Some students claimed they prefer to accept solely textual description of the system, and create the system diagrams on their own. Moreover, graphical representations tend to be ambiguous. Therefore, while diagrams make the whole system easily human interpretable, some assisting textual information is required to prevent misinterpretations.

4.2. Model Multiplicity or Model Singularity?

SysML employs nine types of diagrams, in contrast to the single type of diagram that OPM has. In this part of the forum students were asked to argue about the impact of the number of diagram types on the language comprehension and human communicability. The conclusion of the students' discussion was that homogeneous systems with a well defined hierarchy can be naturally expressed by a single type of diagram with a refinement mechanism. Such a representation benefits from the single context, which remains always the same while traversing the diagrams. However, a complex and flat system can be better modeled using many different types of diagrams, each accounting for some aspect of the system.

4.3. SysML vs. OPM

SysML and OPM are both intended to model general-purpose complex systems. In this part of the forum students compared SysML and OPM modeling languages and concluded in which cases they would prefer each one of the languages.

To illustrate the usability of the two languages, the distiller model implementations example was provided in both OPM and SysML. The students argued that the OPM model was clearer and more communicable than its SysML counterpart. SysML diagrams looked too detailed, making it difficult to grasp the general concepts of the system under study. Students also found the OPM model to be more navigable than the SysML model. Both OPM and SysML support hierarchical organization of information, but the numerous SysML diagrams types caused the students to get lost in the different system views.

The advantages of SysML students found compared with OPM were in expression of detailed scheduled operations. Students also noted that SysML supports textual constraints specified in any appropriate formal language, and this useful feature is missing in OPM.

5. Conclusions

Overall, there is no single “superior” system modeling language. The variety of SysML diagram types enables one to express different aspects of the system. In OPM the well-organized single typed hierarchy of Object-Process Diagrams permits holistic understanding of the system and its environment effortlessly, while concurrently showing both the behavior and the structure of the system. It is harder to express with OPM certain fine points of a complex system, such as those expressible in a SysML Parametric Diagram. On the other hand, SysML diagrams tend to be complicated, requiring time to understand the underlying concepts, which are spread over a number of different views.

Another issue worth noting is the requirements traceability facilities provided by the two languages. The SysML approach is to combine requirements and components using diagrams, where the connection between different system components and the requirements satisfied by them are visualized. In contrast, in OPCAT requirements traceability is done separately from the graphics. The developers monitor connections between OPM design artifacts and the matching requirements through requirements coverage tables that are built into OPCAT. Unlike the SysML requirements/components diagrams, which are likely to be crowded and overloaded with crossing links, OPM deals with the problem by separating traceability information from the graphical representation.

One possible extension of the current research is experimenting with the two languages by using them for modeling systems in different domains and discovering whether, to what extent, and in what ways the specific domain influences the general guidelines for system modeling language selection suggested in this work. Further empirical studies are also needed to analyze the impact of various proportions of textual and graphical representations of a developed model on its comprehensibility and completeness. Finally, defining a hybrid methodology exploiting the advantages of the two languages seems to be a challenging issue.

6. References

[1] UML 2.0 Superstructure Specification (OMG document number formal/05-07-04), available at http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf.

[2] UML 2.0 Infrastructure Specification (OMG document number ptc/05-07-05), available at http://www.omg.org/cgi-bin/apps/doc?formal/05-07-05.pdf.

[3] OMG Systems Modeling Language Specification (OMG document number ptc/06-05-04), available at http://www.omg.org/cgi-bin/apps/doc?ptc/06-05-04.pdf.

[4] OMG Systems Modeling Language, The Official OMG SysML site, http://omgsysml.org/.

[5] Friedenthal S., Moore A., Steiner. R, OMG Systems Modeling Language Tutorial, 11 July 2006, available at http://omgsysml.org/SysML-Tutorial-Baseline-to-INCOSE-060524-low_res.pdf.

[6] Dori D., “Object-Process Methodology- a Holistic Systems Paradigm”, Springer, 2002.

[7] Dori D., “Fostering Meaningful Learning and Argumentation Skills via an Online Forum within a Hybrid Course”, In Proc. AACE E-Learn Conf., Hawaii, 2006.

[8] The Official OPCAT site, http://www.opcat.com.

[9] 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.