

## MIT Open Access Articles

*When quantitative meets qualitative: enhancing OPM conceptual systems modeling with MATLAB computational capabilities*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

**Citation:** Dori, Dov, Aharon Renick, and Niva Wengrowicz. "When Quantitative Meets Qualitative: Enhancing OPM Conceptual Systems Modeling with MATLAB Computational Capabilities." *Research in Engineering Design* 27.2 (2016): 141–164.

**As Published:** <http://dx.doi.org/10.1007/s00163-015-0209-9>

**Publisher:** Springer London

**Persistent URL:** <http://hdl.handle.net/1721.1/103795>

**Version:** Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

**Terms of Use:** Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.



# When Quantitative Meets Qualitative: Enhancing OPM Conceptual Systems Modeling with MATLAB Computational Capabilities

Dov Dori

Technion – Israel Institute of Technology  
Haifa 32000, Israel  
and

Massachusetts Institute of Technology  
Cambridge, MA 02139, USA  
[dori@mit.edu](mailto:dori@mit.edu)

Aharon Renick

Technion – Israel Institute of  
Technology  
Haifa 32000, Israel  
[adrenick@technion.ac.il](mailto:adrenick@technion.ac.il)

Niva Wengrowicz

Technion – Israel Institute of  
Technology  
Haifa 32000, Israel  
[nivawen@technion.ac.il](mailto:nivawen@technion.ac.il)

**Abstract**—Conceptual modeling is an important initial stage in the lifecycle of engineered systems. It is also highly instrumental in studying existing unfamiliar systems—the focus of scientific inquiry. Conceptual modeling methodologies convey key qualitative system aspects, often at the expense of suppressing quantitative ones. We present and assess two approaches for solving this computational simplification problem, defined below, by combining Object Process Methodology (OPM), the new ISO/PAS 19450, with MATLAB or Simulink without compromising the holism and simplicity of the OPM conceptual model. The first approach, AUTOMATLAB, expands the OPM model to a full-fledged MATLAB-based simulation. In the second approach, OPM Computational Subcontractor, computation-enhanced functions replace low-level processes of the OPM model with MATLAB or Simulink models. We demonstrate the OPM Computational Subcontractor on a radar system computation. Experimenting with students on a model of an online shopping system with and without AUTOMATLAB has indicated important benefits of employing this computation layer on top of the native conceptual OPM model.

**Keywords**—*Model-based Systems Engineering, Object-Process Methodology, MATLAB, Simulink, Modeling and Simulation, Conceptual Modeling*

## 1. INTRODUCTION - CONCEPTUAL MODELING

A key stage in the early stages of architecting and design of a new system or in understanding an existing one is its conceptual modeling: creating a primarily-qualitative model, which clearly specifies the system's function (*why* it is designed, what value is it expected to provide its beneficiaries with—the *utility* aspect), its structure (*what* is the system made of, how is the whole related to its parts—the *structural* aspect), and its behavior (*how* the system operates and changes over time and how objects in it are transformed to achieve its function—the *behavioral* aspect). Similarly, when aiming to research and fully understand existing systems, a conceptual

model of the system under study can be highly valuable (Somekh et al, 2014). Different modeling methodologies and languages, such as OPM (Dori, 2002) and SysML (Object Management Group, 2012; Weikens, 2007), enable one to conceptually model a system and simulate its behavior. Holistic understanding is achieved by simplifying certain aspects of the real system, such as its level of detail (Zeigler, 1976). One aspect that is often simplified in conceptual modeling is the computational aspect of a system—the mathematical entities that may govern the actions and reactions of a system, the exact output of some actions, and representation of random effects that determine the dynamics of the system.

While this simplified, qualitative-only view of the system helps in initial, overall comprehension of the system, it often lacks important information, especially when the model is simulated, making it necessary to explicitly express the dynamic, time-varying aspect of the system. Moreover, while making progress in the design or study of a system, simulation of the system's structure and behavior often becomes mandatory to ascertain that the system meets the requirements it is expected to fulfil. This is especially true for systems that exhibit complex behavior, which might require representation of non-deterministic aspects and advanced numerical calculations that drive different actions, as well as sophisticated quantitative decision-making processes. Advancing from the conceptual model to an elaborate simulation is therefore often critical for testing and validating the system under design, or confirming theories regarding systems under study.

In some cases, due to the human in the loop, the transition from the modeling stage to the simulation stage can result in errors or inaccuracies. Creating a simulation of the system can be done by studying the model or the original system directly, aiming to understand it, and building the simulation accordingly. As long as the model-to-simulation transition process involves human intervention, it is prone to mistakes and inaccuracies. Moreover, such manual transitions can

overlook insights gained during early stage of the conceptual model.

## 2. THE COMPUTATIONAL SIMPLIFICATION PROBLEM

We define computational simplification as the simplification of a conceptual model by elimination or reduction of its computational aspects. This abstraction gives rise to the *computational simplification problem*—the problem of lack of complete model comprehension arising from a conceptual model computational simplification. Conceptual systems modeled in some of the modeling languages or methods presented in the following sections, may suffer from this problem. Other methods that enable modeling lower levels, which include computational aspects, lack high-level abstraction abilities, which are vital in conceptual models. The computational simplification problem poses the challenge of equipping system architects, designers, and domain experts with the ability to incorporate computational modules into the conceptual model in order to make it more complete. In this paper we tackle this challenge by presenting and evaluating possible solutions for this problem that expand Object Process Methodology (OPM) with the capabilities of MATLAB and Simulink.

As the design or study of a system progresses, elaborate simulations are often created for examining the system in operation. While in general a human in the loop may introduce errors during the transition from the conceptual model to a simulation, our approach provides for a MATLAB-based simulation, which is created directly from the evolving OPM model, avoiding the likely introduction of new errors, which, given the early stage of their introduction, are often very costly to correct.

## 3. OBJECT PROCESS METHODOLOGY

Object Process Methodology (OPM), the [ISO/PAS 19450](#), titled "*Automation systems and integration – Object-Process Methodology*", is a conceptual modeling approach and language that uses a single unifying bimodal graphical and textual model. This single model captures the functional, structural, and behavioral aspects of a system (Dori, 2002). An OPM model consists of elements that are things and relationships: stateful objects and processes that transform them are the things, which are related by procedural and structural relationships. Objects are the stateful components the system is made of, while processes are things that transform objects by creating or consuming them, or by changing their state.

An OPM model is represented in two complementary modalities: graphical and textual. A set of interconnected Object Process Diagrams (OPDs) constitutes the graphical representation of the model, while a corresponding set of Object Process Language (OPL) paragraphs is the parallel textual representation of the model. OPL is a subset of natural English, which anyone with basic knowledge of English can readily understand. Both representations are completely

interchangeable and convey the same information about the model, appealing to “both sides of the brain” (Dori, 2008).

OPM models can be created with OPCAT (Dori et al., 2010), a CASE tool for designing and testing OPM models. A summary of OPM symbols and rules is presented in appendix A. OPM is specified in Dori (2002; 2015). The ISO standardization process of OPM is described by Blekhman et al. (2011).

## 4. MATLAB & SIMULINK

**MATLAB**<sup>™</sup> (MathWorks, 2011), short for Matrix Laboratory, is a numerical computing environment developed by MathWorks, which is widely used in various fields of engineering and science. MATLAB has powerful built-in library functions that serve a wide variety of uses. Groups of functions for specific applications are collected in packages, referred to as toolboxes. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and many other fields. MATLAB also offers easy graphical command interface, enabling visualization of results immediately and conveniently.

Due to its wide array of function libraries and numerical abilities, MATLAB is commonly employed to generate system simulations, enabling users to perform mathematical operations, such as solving differential equations, implementing stochastic behavior, and manipulating multi-dimensional arrays.

MATLAB has many advantages over conventional programming languages such as C or FORTRAN, enabling it to solve technical problems (Houcque, 2005). MATLAB is an interactive system whose basic data element is an array that does not require dimensioning, so the user can add and modify elements dynamically as the program proceeds, without defining them in advance. The software package has been commercially available since 1984 and is now considered a standard tool at most universities and industries worldwide. While the commercial version is commonly used, there are some free and open source MATLAB-compatible solutions, including Octave, Scilab, and FreeMat (Sharma & Gobbert, 2010).

Simulink, a platform incorporated within MATLAB (Mathworks, 2008), enables programming by means of a graphical display – dragging and connecting predefined blocks, placing them in different places, masking them, and manipulating them. Moreover, Simulink introduces an ability to incorporate Stateflow modules in a system simulation. Simulink’s main addition to MATLAB is its graphical environment that replaces the textual code-based view. Similar to the original MATLAB environment, Simulink provides a set of block libraries that are analogous to the MATLAB library functions. Simulink has many block libraries, including Math Operations, Model Verification, Ports & Subsystems, Signal Routing, Sinks, and Sources. It is also possible to define blocks by programming with MATLAB code. Simulink’s graphical environment can serve as a means to model a system, leveraging on the computational strength of

MATLAB. For starters, one can build a simulation of a system represented in MATLAB by using Simulink, adding a graphical representation to the simulation.

Although Simulink enables a graphical approach to the simulation, it does not provide a complete solution to conceptual modeling. Rather, it provides a graphical view of the MATLAB code. The Simulink environment is limited to a block diagram approach, representing each section or subsystem as a separate block (box), which can be drilled into, and can contain lower level subsystems as separate boxes. This approach provides a structural view of a system, but it does not capture the dynamics of the system, nor does it represent relationships between the system and the environment any more than a code-based simulation.

## 5. CONCEPTUAL MODELING LANGUAGES AND METHODS

### 5.1. UML and SysML

The Unified Modeling Language (UML) (Object Management Group, 2011) is an object-oriented modeling language that became the Object Management Group (OMG) standard for software systems development in 1997. UML consists of a model with 14 different views, represented by different kinds of diagrams, many of which evolved from diagrams that were in use during the early 1990's. The fourteen UML views are activity diagram, class diagram, communication diagram, component diagram, composite structure diagram, deployment diagram, interaction overview diagram, object diagram, package diagram, profile diagram, sequence diagram, state diagram, timing diagram, and use case diagram. These 14 views aim to convey different aspects of the system, its structure, behavior, relationships, and change over time, so that a system modeled by UML consists of related diagrams of different kinds.

SysML, Systems Modeling Language (Object Management Group, 2012) is a profile of UML designed to be more system-centric, as opposed to the more software-centric design of UML. SysML has retained seven of the 14 UML diagram types, modifying some of them, and added two new ones—requirements diagram and parametric diagram.

Comparing UML and SysML on one hand with OPM on the other hand, a major difference that sticks out is the holism of the model. UML and SysML require using at least a subset of the 14 (in UML) or nine (SysML) diagram kinds to represent the system. Conversely, OPM represents the system with an interconnected hierarchical set of diagrams of a single kind—Object-Process Diagram (OPD). This graphical representation is accompanied by an equivalent set of Object-Process Language (OPL) paragraphs that specifies the system in a subset of natural English. Following this minimalism principle, we aim to extend the computational power of OPM while minimizing additional diagram kinds.

### 5.2. Modelica

Modelica (Mattsson & Elmqvist, 1997) is an object-oriented equation-based modeling methodology. The Modelica

language defines and describes the system, its components, structure, and behavior by a set of mathematical equations. The Modelica methodology includes CASE tools for designing Modelica models. These tools allow the user to draw or import a scheme of the system, connecting the model equations to the appropriate component on the scheme. Using Modelica, one can decompose the model hierarchically, simplifying the model and making it more understandable. Another feature of Modelica is a large selection of model libraries, offering predefined subsections of systems, sorted into different fields (electrical, mechanical, aerospace, etc.), which can be easily implemented as part of a model. Other predefined libraries enable integration of numerical solutions and stochastic behavior modules.

Modelica supports quantitative and stochastic aspects due to its equation-based representation, allowing direct modeling of the computational aspects of the system. Describing the architecture of the modeled system using Modelica without the equation-based representation is also possible. However, describing the behavior of the system does require using the mathematical representation, demanding a greater level of user effort to comprehend the model in comparison to a simpler representation of a system flow. Another somewhat limiting aspect is that Modelica definitions are object-oriented, confining the users to think in terms of the OO paradigm, which emphasizes objects and system structure at the expense of suppressing behavioral aspects and processes (“methods”) as secondary to and owned by objects.

### 5.3. Play-in/Play-out

Play-in/Play-out (Harel and Marely, 2003) is an approach for modeling systems, specifically the reactions of the system with the environment. Play-in/Play-out scenarios are “played in” by the user modeling the system using the “Play-engine.” Using an intuitive graphical user interface (GUI) that represents the system, the modeler executes the various actions that affect the system and its expected reactions. As the system behavior is played in, the play-engine automatically generates a set of Live Sequence Charts (LSCs), specifying the behavior of the system.

After creating LSCs that cover the system's permitted actions and their result (and possibly forbidden actions), the model can be “played out.” In the play-out mode, the user can apply an action to the GUI and it will react according to the set of rules defined by the LSCs. Playing out different scenarios can help identify contradictions, undefined actions, and other errors.

The play-engine allows specifying an event value as a function that is predefined in the GUI code (using, for example, Visual Basic). This option enables modeling complex systems, where the result of an action is not as simple as a constant reaction, but rather has quantitative aspects. The play-engine has also a limited support of non-deterministic actions; one can define more than a single possible reaction for each played-in action and allocate a probability to each reaction.

While the Play-in/Play-out approach might seem to satisfy the requirement of handling quantitative and stochastic aspects, this is only partially correct, since this method is geared for scenario-based modeling, emphasizing the behavioral aspects of the system. The ability to model stochastic behavior with the play-engine is limited, as non-uniform continuously distributed probabilities and other forms of stochastic behavior are difficult to define in a straightforward manner. While it is possible to predefine functions as part of the GUI code in a visual programming environment, the Play-in/Play-out method lacks ability to directly incorporate quantitative aspects into the model. Conversely, MATLAB enables direct access to arrays and many toolboxes that provide for incorporating quantitative and non-deterministic aspects.

#### 5.4. *MLDesigner*

MLDesigner (Schultz et al., 2010) is an open environment for UNIX or Linux systems used for designing and testing of system architectures and their functions. An MLDesigner model consists of a block diagram containing a variant of C++ code that represents the functionality of each block. Blocks can be modified by editing their size, position, color, etc. Many predefined code blocks are available, simplifying the design and execution of a model. MLDesigner can partially serve as a computational layer of a conceptual modeling language, similar to one of the solutions suggested in this work. A work in this direction has been presented by Schultz et al. (2010), where an OPM-to-MLDesigner translation was proposed to add simulation capabilities to an OPM model. The main advantage of the OPM-MLDesigner approach is the simulation abilities that are built into the MLDesigner CASE tool, which, for simple simulations, can be more convenient than MATLAB. The OPM-MLDesigner approach has some downsides: MLDesigner is not as widespread as MATLAB and it is not Windows-compatible. The OPM-MLDesigner approach aims to generate an MLDesigner model from the OPM model by using OPL—the textual modality of the OPM model—to generate an MLDesigner model, referred to as MML. MML is not linked to the OPM model and OPCAT simulation tool, and it does not affect the original OPM model. In other words, the OPM-MLDesigner approach does not improve OPM’s computational capabilities, but rather generates the OPM model in a different language. As MLDesigner focuses on processes, it does not have an entity compatible with the OPM concept of object, making it necessary to define dummy processes that play the role of objects. This can substantially complicate the model.

#### 5.5. *Arena*

Arena (Kelton et al., 2014) is a widely used, general purpose simulation tool, which enables the user to build a visual model of the system using a graphic editor, which is converted into the SIMAN language. The modules of a system are represented as boxes of different shapes, while lines represent connections between modules and actions. The flow and

timing of each entity in the system is defined from its arrival to its departure. Arena supports common functions and mathematical operators, and allows the user to define new models and behaviors by using external solutions, such as Visual Basic.

An approach to designing simulations using OPM was presented by Gilat (2002). This research focused on using OPM to better specify and design a model of a given simulation in order to improve its design and clarity. As a case study, the approach was demonstrated using the Arena simulation tool, where the advantages of connecting Arena and OPM in various scenarios were shown in experiments. In the Arena-OPM approach, the advantages of conceptual modeling are utilized only during the simulation design stage, but not in the simulation creation and operation stages. Since we aim to improve both the model and its simulation by incorporating the quantitative aspects into the model itself, the Arena-OPM approach does not satisfy the requirement of a combined conceptual and computational modeling methodology.

#### 5.6. *Multi-Representation Architecture*

Peak et al. (1998) proposed a Multi-Representation Architecture (MRA) approach to integrating computer-aided design (CAD) with computer-aided engineering (CAE), which maps design and analysis models onto each other. They used four information representations. One is the product model—the master description of the product being designed, which uses design tools. The three other representations are analysis models that use solution tools at increasing levels of abstraction. Our approach is different in that we focus initially on the very early conceptual design phases, seeking for a concept for a problem of fulfilling a required function that delivers value to a beneficiary before delving into the technical complexities of the particular solution.

#### 5.7.A *computational product model for conceptual design using SysML; Improving PDM systems*

Wölkl and Shea (2009) proposed a computational product model for conceptual design using SysML. Their proposal moves beyond geometry to integrating all the necessary aspects for conceptual design. Using a case study on the design of a passenger car’s luggage compartment cover, they show that many different SysML diagram kinds are suitable for formal modeling in mechanical concept design. Accordingly, they proposed the creation and use of libraries defining generic templates that raise efficiency in modeling. Since SysML contains nine types of diagrams and OPM only one, the complexity of the integrated approach using OPM as the underlying conceptual modeling language is smaller. Bergsjö et al. (2010), who investigate the need to formalize the systems engineering processes for embedded systems at a large automotive company, noted that formalization could be developed by using a commercial Product Data Management

(PDM) system. They found that the company had an informal work process that delegated responsibility to individual design engineers with low level of enforcing formal reports and deliverables. Based on this finding, they customized the PDM system to work for two demonstrator cases focusing on integration and workflow using Visual C# to automate the design task in conjunction with the PDM system and simulation software.

Indeed, a major obstacle in the lifecycle engineering of a product or a system is the bumpy transition from conceptual, qualitative high-level abstract architecture and design to detailed, quantitative analysis of the design to ensure the systems resilience, robustness, and other required features. Our approach of integrating the qualitative high-level conceptual stage with the quantitative one that follows it can be considered a step in solving this problem.

## 6. RESEARCH MOTIVATION AND GOALS

A basic requirement of a modeling and simulation language is expressiveness that provides for modeling the phenomena encountered in the design of a system, such as non-linear, multi-disciplinary, continuous or discrete flows. The models must also be easy to create and reuse (Sinha et al., 2001). Simplification of the computational aspect of a system, which provides an accurate, quantitative representation of the system’s behavior, might conceal some important information about the system. Certain OPM models might be adversely affected by this computational simplification problem.

Many of the modeling languages and methods presented in the previous section provide conceptual models of the system under development or study and enable some kind of simulation. However, in most cases, either the level of the quantitative aspect of the simulation is insufficient, or it is achieved at the expense of sacrificing simplicity and generality of the method and the resulting model.

MATLAB is a convenient tool for simulating complex systems, but it does not have advanced abstract conceptual modeling capabilities. Adding a numerical computational layer to the conceptual modeling power of OPM provides for simulating its behavior both qualitatively and quantitatively.

While OPM enables exporting a model as an XML file, JAVA code and more, it does not fully integrate into common development tools such as MATLAB. Exporting a MATLAB-based representation of the OPM model allows one to express the model so that a domain expert who uses MATLAB can understand the system without knowledge of OPM.

A conceptual model must not be overly complex. In other words, “The conceptual modeling mantra is one of developing the simplest model possible to meet the objectives of the simulation study” (Robinson, 2010). Figure 1 illustrates how excessive levels of detail hamper model accuracy. At a certain point, the increasing level of detail reduces the model accuracy due to lack of knowledge on how to model such details. It is thus clear that we would want a model to be as simple as possible while still allowing sufficient accuracy and detail required for the model objective.

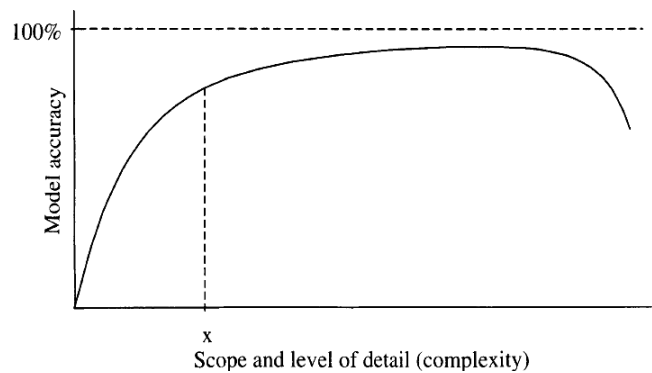


Figure 1: Model and simulation accuracy vs. complexity (Robinson, 2008)

Although simple models are desirable, a simpler model requires larger assumptions regarding the system, risking the possibility of missing important aspects of the system (Davies et al., 2003). Modeling a system with a solution such as MATLAB or Simulink, without OPM, might thus be problematic. Incorporating MATLAB and Simulink representations, where required, into OPM enables model simplification while avoiding the risk of over-simplification. In view of this need, our research aims to alleviate the computational simplification problem and bridge the gap between the qualitative and the quantitative model aspects by enhancing OPM with a MATLAB or Simulink-based computational layer. We present and compare two approaches: AUTOMATLAB—an OPM MATLAB Layer, and OPM Computational Subcontractor.

## 7. AUTOMATLAB

In the AUTOMATLAB approach, a layer of MATLAB is added on top of the OPM model. The OPM model of the AUTOMATLAB-Based Simulating<sup>1</sup> system, shown in Figure 2 (the graphical modality – Object-Process Diagram, OPD) and in Figure 3 (the corresponding auto-generated textual modality – Object-Process Language, OPL, paragraph), consists of three main processes:

1. **MATLAB Code Generating**, which uses the graphical (OPD) or textual (OPL) representation of the **Original OPM Model**. The **Simple MATLAB Code** resulting from this stage specifies the same system as the **Original OPM Model** does.
2. **MATLAB Code Enhancing**, where the user can add predefined standard numerical OPM model snippets to enhance computational aspects of interest as code in the MATLAB file, creating the **Enhanced MATLAB Code**.
3. **Enhanced Model Simulating**, in which the OPM model, enhanced with MATLAB code—the **Enhanced MATLAB Code**—is simulated in OPCAT, resulting in the **MATLAB-Enhanced OPCAT Simulation**.

<sup>1</sup> Boldface names indicate OPM model names of objects and processes.

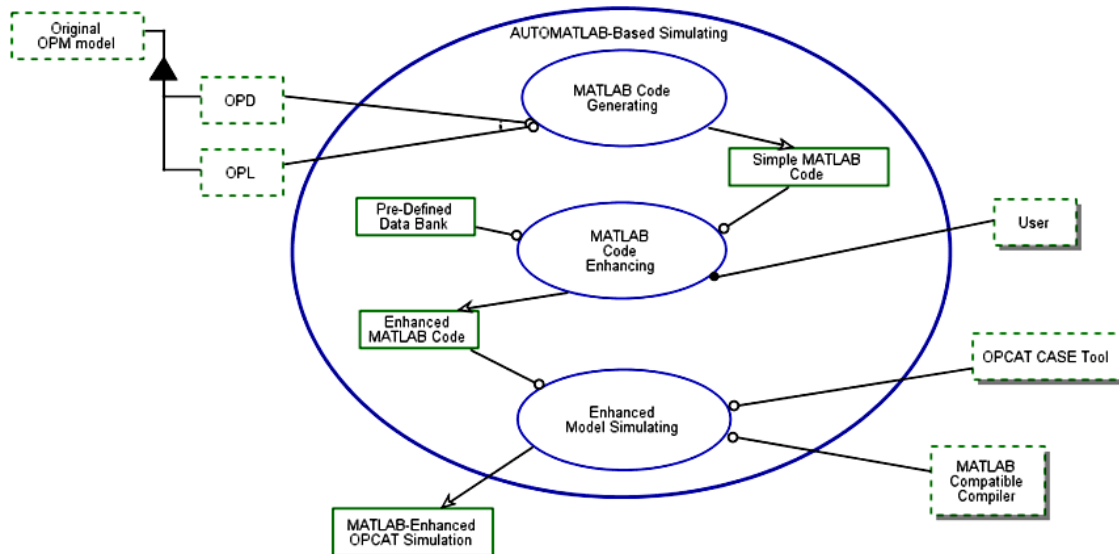


Figure 2: OPM model of AUTOMATLAB – the graphic modality, Object-Process Diagram (OPD)

AUTOMATLAB-Based Simulating consists of MATLAB Code Generating, MATLAB Code Enhancing, and Enhanced OPCAT Simulating.

AUTOMATLAB-Based Simulating exhibits Simple MATLAB Code, Enhanced MATLAB Code, and Pre-Defined Data Bank.

AUTOMATLAB-Based Simulating zooms into MATLAB Code Generating, MATLAB Code Enhancing, and Enhanced OPCAT Simulating, as well as Pre-Defined Data Bank, Enhanced MATLAB Code, and Simple MATLAB Code.

MATLAB Code Generating requires either OPD or OPL.

MATLAB Code Generating yields Simple MATLAB Code.

MATLAB Code Enhancing requires Simple MATLAB Code and Pre-Defined Data Bank.

MATLAB Code Enhancing yields Enhanced MATLAB Code.

Enhanced OPCAT Simulating requires Enhanced MATLAB Code, MATLAB Compatible Compiler, and OPCAT CASE Tool.

Enhanced OPCAT Simulating yields MATLAB-Enhanced OPCAT Simulation.

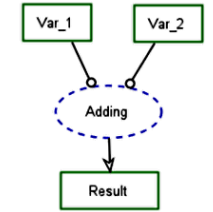
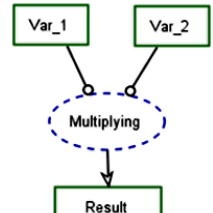
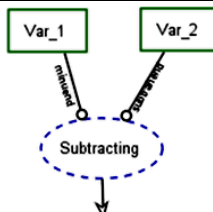
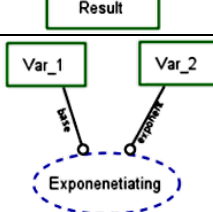
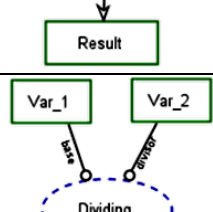
Original OPM model consists of OPL and OPD.

User handles MATLAB Code Enhancing.

In order to generate the AUTOMATLAB layer efficiently and accurately during the MATLAB Code Generating stage, we define the syntax and semantics of the various OPM constructs and their MATLAB translations. This enables the modeler to add to the original OPM model the computational aspects expressed in OPM. As a first step, we have mapped the main basic built-in MATLAB functions in the MATLAB documentation (MathWorks, 2011) to their OPM equivalents. To distinguish these built-in functions from user-created functions that may have different meanings, the process names listed in Table 1 through Table 4 are added to the list of OPM reserved words.

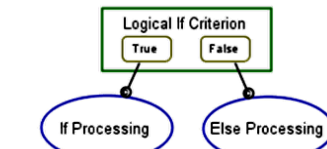
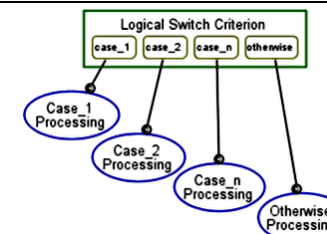
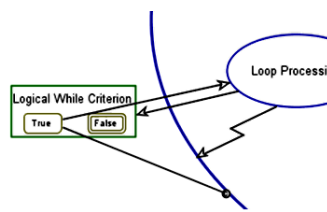
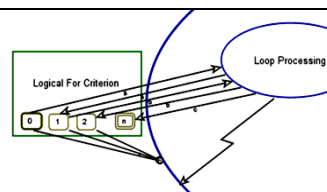
Figure 3: OPM model of AUTOMATLAB – the textual modality, Object-Process Language (OPL) paragraph that is compatible with and was automatically generated from the OPD in Figure 2

Table 1: Examples of AUTOMATLAB arithmetic operators

Symbol	Operator OPM Process Name	OPD
+	Addition <b>Adding</b>	
*	Multiplication <b>Multiplying</b>	
-	Subtraction <b>Subtracting</b>	
^	Exponentiation <b>Exponentiating</b>	
/	Division <b>Dividing</b>	

In order to minimize changes to the OPCAT tool, we generate the code from an html file of the OPM model expressed in OPL and exported from OPCAT. Processes, objects, values, and relationships are identified from the OPL sentences. OPM constructs are mapped into three matrices: process-to-process relationships, object-to-object relationships, and process-to-object relationships.

Table 2: Examples of AUTOMATLAB loops and control structures

Operator / Process Name	OPD
if then...else	
switch	
While	
for	

Each relationships is then translated into the appropriate code segment in a separate MATLAB file called *m file*. For example, as the first line in Figure 4 shows, a ‘requires’ relationships between a process A and object B means that B is instrument for executing A, so in order for A to execute, B must be present. This semantics is translated to the MATLAB code segment in the m file listed in lines 2-4 of Figure 4.

```

1      % A requires B.
2      if ~isempty(B)
3          A();
4      end

```

Figure 4: The MATLAB code for the OPL sentence “A requires B.”

When other relationships are translated, the code will change accordingly. For example, suppose that in addition to the example in Figure 4, a ‘yields’ relationship exists between process A and object C, which means that C results from executing A. In that case, the code in Figure 4 will be altered to the one shown in Figure 5.



```

1  % A requires B.
2  ~ if isempty(B)
3  ~ [C] = A();
4  ~ end

```

Figure 5: The MATLAB code for the OPL sentences “A requires B.” and “A yields C.”

Conveying only the information represented the original OPM model, the MATLAB code generated by AUTOMATLAB is simple and readily executable. When the OPM model does not have the information needed to complete a legal code segment, that piece is left commented. For example, we noted that the process A requires B, but as long as A is an atomic process, i.e., there is no in-zoomed (drilled-down) view of the process A specifying what A “does”, the use of B in function A remains as a comment, as shown in line 5 in Figure 6. In a similar way, all the process-to-process relationships (consists of, zooms into, etc.), object-to-object relationships (consists of, exhibits, is a, etc.) and process to object relationships (consumes, requires, changes, etc.) are translated into corresponding code segments that convey the same semantics as their OPM counterparts.

```

1  function [C] = A(B)
2  %A requires B
3  %A yields C
4
5  % B; %A requires B
6
7  C = 1; %A yields C

```

Figure 6: The MATLAB code for the OPL sentences “A requires B.” and “A yields C.” when A is atomic (does not have an in-zoomed view)

Table 3: AUTOMATLAB trigonometric & exponential functions

Operator / Process Name	Description	OPD
acos	Inverse cosine; result in radians	
asin	Inverse sine; result in radians	
atan	Inverse tangent; result in radians	
cot	Cotangent of argument in radians	
sin	Sine of argument in radians	
tan	Tangent of argument in radians	
exp	Exponential	
log	Natural logarithm	
log10	Common (base 10) logarithm	
sqrt	Square root	

Table 4: AUTOMATLAB miscellaneous functions

Operator / Process Name	Description	OPD
abs	Absolute value	
factor	Returns a row vector containing the prime factors of input.	
fft	Returns the discrete Fourier transform (DFT) of vector x, computed with a fast Fourier transform (FFT) algorithm	
isempty	Determine whether array / variable is empty (skips block of code if is empty)	
rand	Uniformly distributed pseudorandom numbers	
size	returns the sizes of each dimension of array	
randn	Normally distributed pseudorandom numbers	

### 8. OPM COMPUTATIONAL SUBCONTRACTOR

One of the drawbacks of the AUTOMATLAB approach presented above is that it allows the violation of the OPM semantics by manipulating the MATLAB code in a way that

renders the OPM model illegal. This implies that in order to utilize the AUTOMATLAB approach correctly, one needs to master the OPM semantics. To overcome this problem, we have developed the OPM MATLAB Computational Subcontractor (OPM/CS) as an alternative solution to AUTOMATLAB for solving the computational simplification problem.

In the OPM/CS architecture, MATLAB or Simulink act as a "computational subcontractor" for the OPM model simulation: Any process in an OPM model can now be in-zoomed to expose a MATLAB code piece or a Simulink diagram instead of the usual new self-similar OPD resulting from zooming into a process. The OPM simulation execution runs normally until it reaches a process that was in-zoomed by the computational subcontractor and therefore contains a MATLAB code piece or a Simulink diagram. At this point, the input to the process – the existing objects and their states – are sent to the MATLAB function or to the Simulink diagram, and the sub-simulation function is called. The outcome of this sub-simulation defines the outcome of the in-zoomed process, which might include newly created objects, the state of an object upon exiting the process, or a value of an attribute, and the OPCAT simulation continues accordingly.

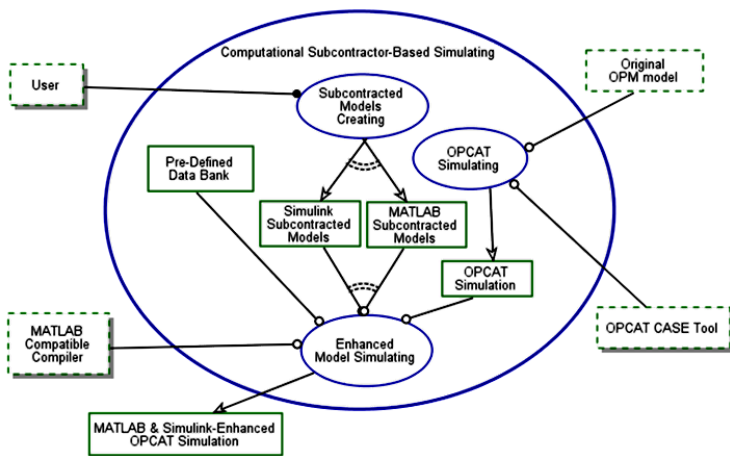


Figure 7: OPM Model of the OPM Computational Subcontractor-Based Simulating system. Top: OPD. Bottom: Corresponding OPL paragraph

If the subcontracted process is a commonly used mathematical function, the user does not need to create it in MATLAB or Simulink, but rather mark it with the notation <<CS>> (short for Computational Subcontractor). The in-zoomed process will then contain the predefined function called from the library presented in Table 1 through Table 4. In this case, the user does not need to have any knowledge of MATLAB or Simulink to use it as computational subcontractors. If the process is in-zoomed by a user-defined function or diagram, the process is linked to the appropriate MATLAB function or Simulink diagram by specifying the folder in which the user-defined function or diagrams are saved, using the same name for both the OPM process and the MATLAB function or Simulink diagram.

Computational Subcontractor-Based Simulating exhibits MATLAB Subcontracted Models, Pre-Defined Data Bank, Simulink Subcontracted Models, and OPCAT Simulation. Computational Subcontractor-Based Simulating consists of Subcontracted Models Creating, Enhanced Model Simulating, and OPCAT Simulating. Computational Subcontractor-Based Simulating zooms into Subcontracted Models Creating, OPCAT Simulating, and Enhanced Model Simulating, as well as OPCAT Simulation, Simulink Subcontracted Models, Pre-Defined Data Bank, and MATLAB Subcontracted Models. Subcontracted Models Creating yields Simulink Subcontracted Models or MATLAB Subcontracted Models. OPCAT Simulating requires Original OPM model and OPCAT CASE Tool. OPCAT Simulating yields OPCAT Simulation. Enhanced Model Simulating requires OPCAT Simulation, MATLAB Compatible Compiler, and Pre-Defined Data Bank. Enhanced Model Simulating requires Simulink Subcontracted Models or MATLAB Subcontracted Models. Enhanced Model Simulating yields MATLAB & Simulink-Enhanced OPCAT Simulation. User handles Subcontracted Models Creating.

Figure 8: The automatically-generated text (in OPL – Object-Process Language) that was created by OPCAT from the Object-Process Diagram (OPD) in Figure 7

In contrast to AUTOMATLAB, the OPM semantics in OPM/CS cannot be breached, since each called MATLAB function or Simulink diagram exists only within the scope of a single in-zoomed process, and it can only affect objects within that scope. To ensure this, the use of global variables in the subcontracted functions is not allowed.

OPM/CS contains three parts: the OPM model in OPCAT, The MATLAB or Simulink models library, and an OPM/CS Manager. The subcontractor manager supports both MATLAB and Simulink as alternative subcontractors and the predefined common functions in Table 1 through Table 4. The OPM/CS Manager controls the bidirectional transactions between the OPM and the MATLAB/Simulink environments. An OPD describing the Architecture of OPM Computational Subcontractor is presented in Figure 7. Figure 8 presents the automatically-generated text (in OPL – Object-Process Language) that was created by OPCAT from the Object-Process Diagram (OPD) in Figure 7. The OPM/CS manager runs in the MATLAB environment, and it calls the MATLAB functions (expressed as m files) or Simulink diagrams (expressed as mdl files) directly from this environment.

### 9. RADAR SEARCHING & TRACKING: A CASE STUDY

We present a case study that demonstrated the use of OPM/CS for a search and tracking radar system. The OPM model of this system is presented in Figure 9. The Searching process creates a Detection output, which can be either 0 (no detection) or 1 (positive detection). If detection is achieved, the Tracking process tracks the Target and creates a Track Output, and when this process ends, either Detection Details Message or No Detection Message is created.

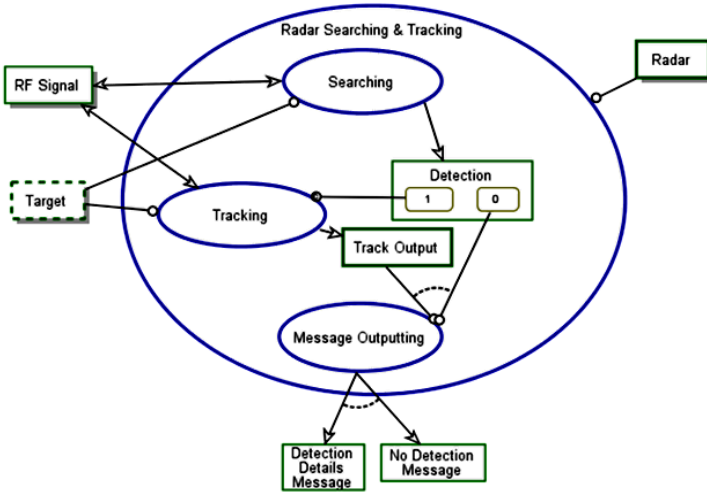


Figure 9: Radar Searching & Tracking in-zoomed

In the in-zoomed **Searching** process in Figure 10, we see that it consist of two subprocesses: **Transmitting** and **Receiving & Processing**. **Transmitting** requires the radar characteristics **Transmitting Power [W]**, **Gain TX [dB]**, and **Wavelength [m]**. **Transmitting** creates the **RF signal**. The process **Receiving & Processing**, which requires the **Target** attributes and some of the **Radar** attributes, consumes the **RF signal**.

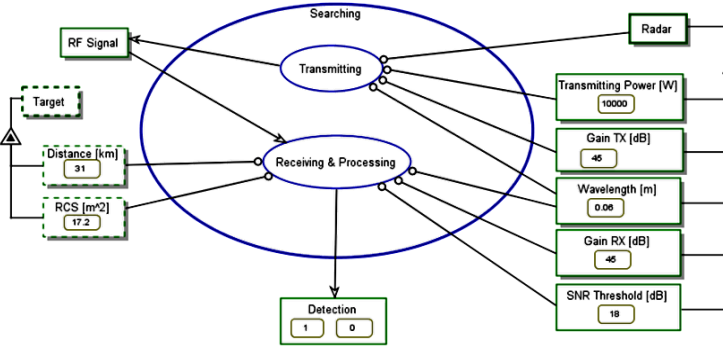


Figure 10: SD1.1 - Searching in-zoomed

When running the OPM simulation of the model shown in Figure 9, **Searching** creates detection in either the **1** state or the **0** state randomly, regardless of the **Target** and **Radar** attribute values.

The received power of a target is described by the Radar Equation 1 (Scolnik, 1980).

$$P_r = \frac{P_t G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4 L} \quad (1)$$

In most cases, the Radar Equation is written in log 10 form ( $10 \cdot \log_{10}(X)$ ), taking the following form in Equation 2.

$$P_r = P_t + G_t + G_r + \sigma + 2\lambda - 3 \cdot 4\pi - 4R - L \quad (2)$$

Here,  $P_r$  is the received power,  $P_t$  is the transmitted power,  $G_t$  is the transmitter gain,  $G_r$  is the receiver gain,  $\sigma$  is the target radar cross-section,  $\lambda$  is the radar wavelength,  $R$  is the target

distance, and  $L$  represents radar losses. If the ratio (or difference, in the log form) between  $P_r$  and the level of noise is above the signal-to-noise threshold, the target is detected.

An OPM model of the Radar Equation implementation for **Searching** is complex and unintuitive due to OPM lack of ability to represent formulae in a simple form. To visualize this unnecessarily complicated view of the OPM model when it comes to relatively simple equations, in **Error! Reference source not found.** we show the OPM model of **Searching** that implements the Radar Equation, where the definitions for addition, subtraction, multiplication, common log and the control structure 'if then...else' are taken from Table 1 through Table 4.

Evidently, this OPM model for representing the relatively simple radar equation is complicated. The in-zoomed content of **Searching** can be replaced by the simpler MATLAB code in Figure 11 or the corresponding Simulink diagram in Figure 12, although each of them is also more complicated than the Radar Equation (1), which is probably the most compact expression, as it uses the mathematical conventions of multiplication (lack of any symbol), exponentiation (as superscript), parentheses, etc.

```

1 function detection = Searching(transmitting_power_W, gain_TX_dB, gain_RX_dB, ...
2                               RCS_m2, wavelength_m, distance_km, SNR_threshold_dB)
3
4 Loss = 3; %[dB];
5 noise = -120; %[dBm]
6
7 % Pr [dBm]
8 Pr = ...
9 10*log10(transmitting_power_W) + gain_TX_dB + gain_RX_dB...
10 + 10*log10(RCS_m2) + 20*log10(wavelength_m)...
11 - 10^3*40*log10(distance_km) - 4*30*log10(pi) - Loss;
12
13 if (Pr - noise) > SNR_threshold_dB
14     detection = 1;
15 else
16     detection = 0;
17 end

```

Figure 11: MATLAB code of the simple Radar Detection condition

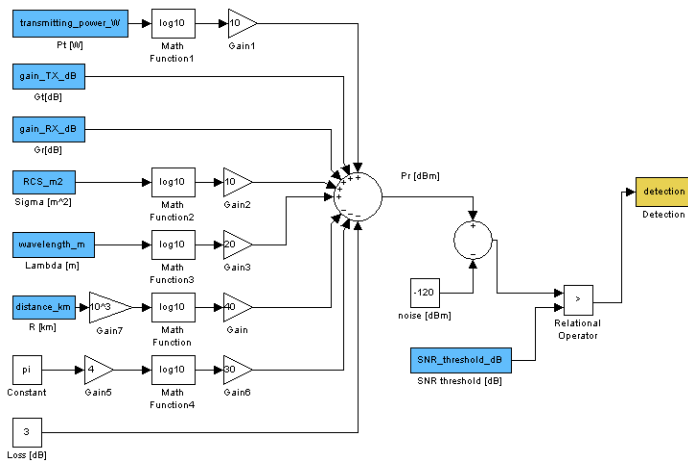


Figure 12: A Simulink diagram of the Radar Detection condition

Applying OPM/CS, the OPCAT simulation arrives at the **Searching** process. Then, the MATLAB code (Figure 11) or the Simulink diagram (Figure 12) is called. It reads the data files containing the inputs to the diagram, calculates the outcome, and returns the appropriate output using the data files.

A demonstration of a successful detection, based on computing  $P_r$  in the Radar Equation, is shown in Figure 14 through Figure 17. An important aspect of using MATLAB or Simulink as a subcontractor for the OPM model is the ability to change the level of complexity of the Simulink model without changing the OPM model itself. For example, we can replace the simple Simulink model in Figure 12 with the more complex model presented in Figure 18. This more elaborate model can be used instead of the model in Figure 12 without altering the OPM model whatsoever, providing for further simulation of the radar systems according to the designers' needs.

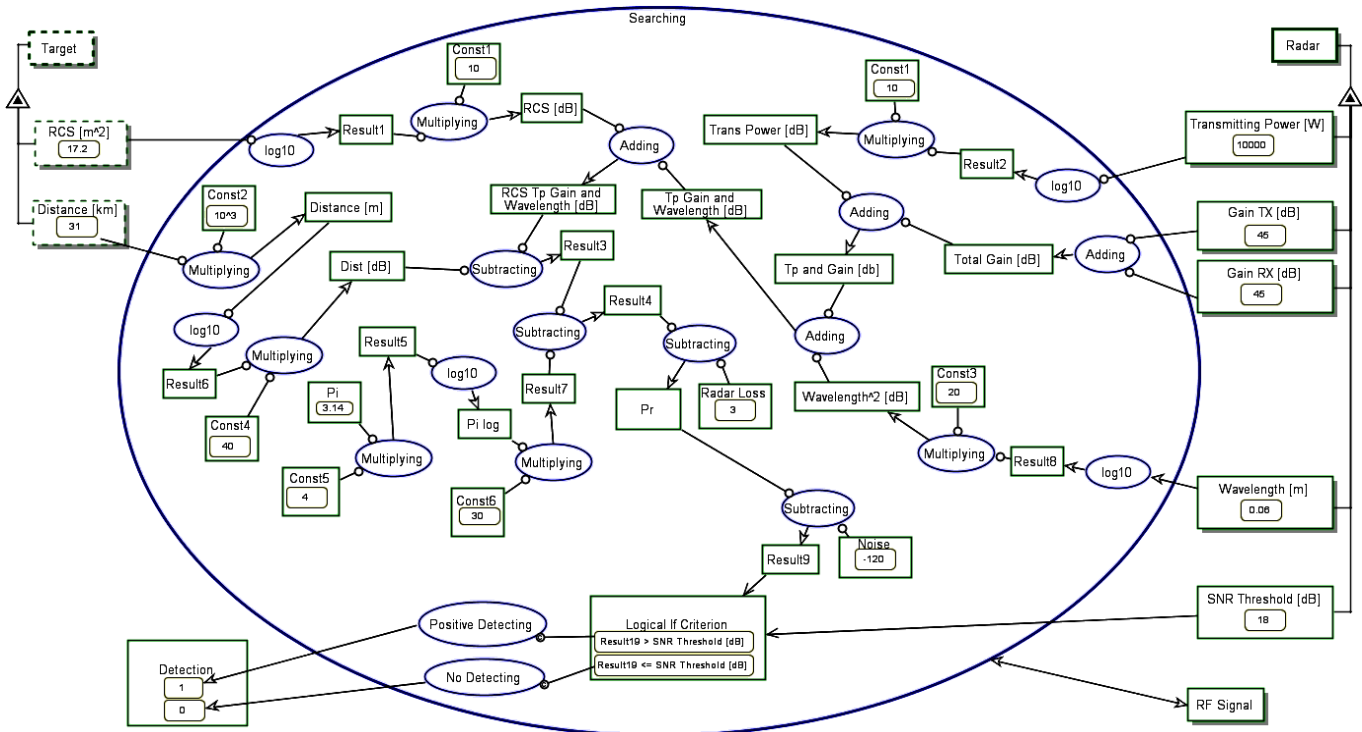


Figure 13: OPM model of **Searching** that implements the Radar Equation demonstrating that at the computational level mathematical expressions and MATLAB is preferable

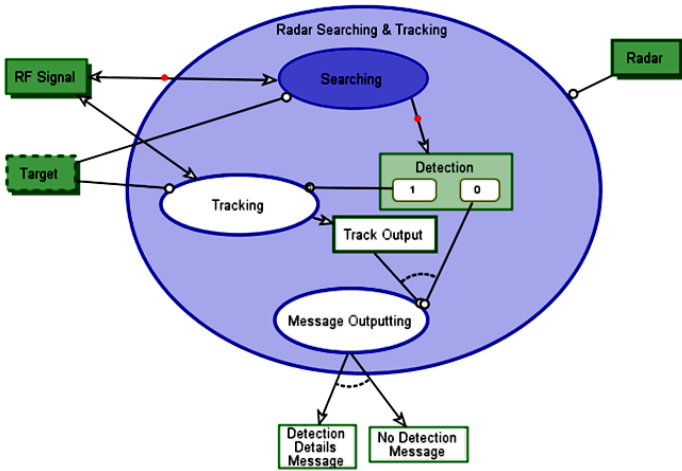


Figure 14: The MATLAB-enhanced OPCAT simulation: The OPCAT simulation calls the subcontracted **Searching** MATLAB function.

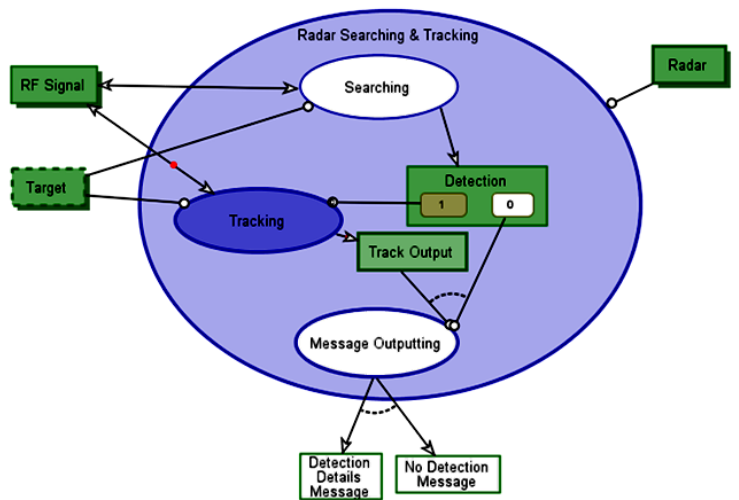


Figure 17: The outcome of the subcontracted **Searching** function in MATLAB with **Detection** value set to 1, causing the OPCAT simulation to proceed accordingly.

```

Op2Mat.txt
1 1
2 *
3 Searching
4 *
5 transmitting_power_W = "10000"
6 gain_TX_dB = "45"
7 gain_RX_dB = "45"
8 RCS_m2 = "17.2"
9 wavelength_m = "0.06"
10 distance_km = "31"
11 SNR_threshold_dB = "18"
12 *
13 detection = {"0","1"}

```

Figure 15: A call to the subcontracted **Searching** function with its input values as an example of a message sent from OPCAT to MATLAB

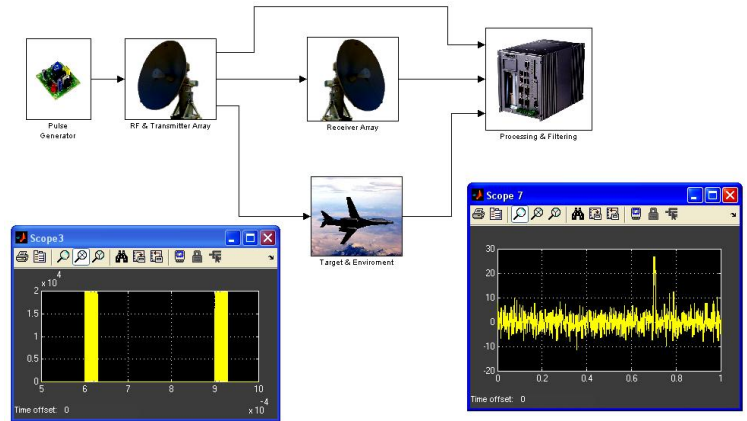


Figure 18: A Simulink diagram of the full Radar simulation, Radar modulated pulse signal (left scope) and SNR graph with positive detection (right scope)

```

Mat2Op.txt
1 1
2 *
3 Searching
4 *
5 detection = {"1"}

```

Figure 16: The outcome message of the subcontracted function **Searching** with **Detection** value set to 1 returned from MATLAB to OPCAT

## 10. EVALUATION OF AUTOMATLAB WITH HUMAN SUBJECTS

An evaluation of the AUTOMATLAB approach was conducted with human subjects in order to more thoroughly assess its benefits and outcomes. The evaluation took place during the spring semester 2013, at the Technion, Israel Institute of Technology. A total of 12 undergraduate students from the Faculty of Industrial Engineering and Management who participated in the course *Specification and Analysis of Information Systems* took part in the evaluation at the end of the semester.

All students (N=12) had knowledge of OPM, as it was taught throughout the semester as part of the course. Some students (N<sub>1</sub>=5) had prior knowledge of MATLAB from previous courses or work, while the rest (N<sub>2</sub>=7) had none or very little knowledge of MATLAB. The students with prior knowledge of MATLAB were the experimental group, while the rest served as the control group. In order to extend our sample,

each student performed the evaluation for two different data sets, achieving a total of  $\tilde{N}=24$ , with  $\tilde{N}_1=10$  and  $\tilde{N}_2=14$ .

The evaluation was based on an OPM model of a **Web Based Grocery Shopping** system, shown in Figure 19 through Figure 22, which had been created by students in the course. This model, rather than a more technically-oriented model as the radar Equation, was selected in order for students to focus on the modeling of a familiar example from their daily lives and not be distracted by the need to learn new material in physics or engineering. The students who were not part of the group that had created the model were briefed about the system and its model.

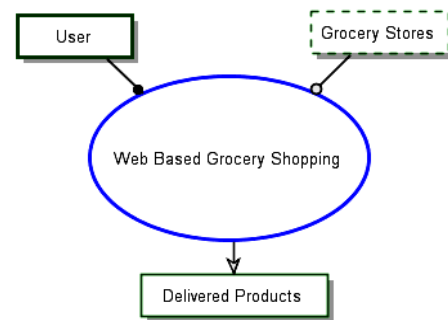


Figure 19: Top-level, system diagram (SD) of the **Web Based Grocery Shopping** system

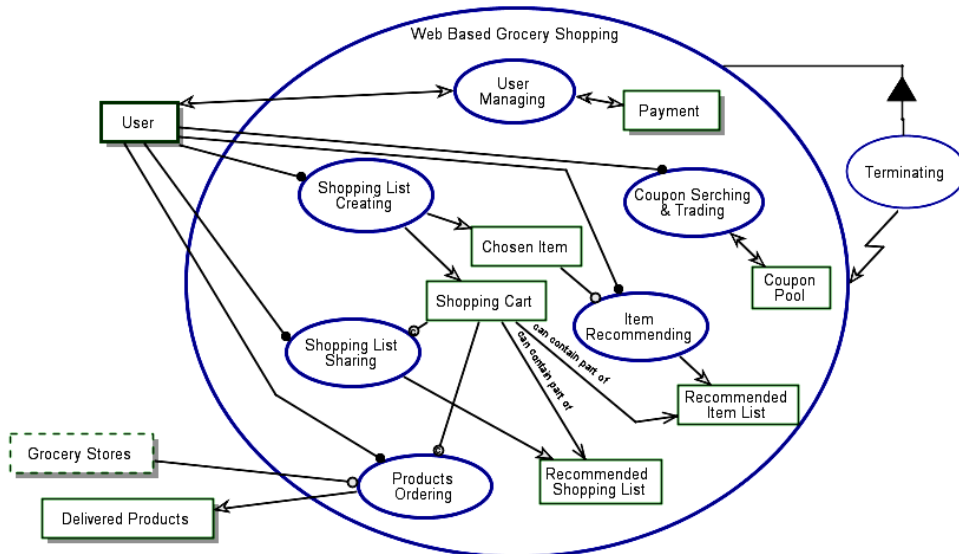


Figure 20: SD1 - **Web Based Grocery Shopping** in-zoomed

The students were requested to analyze some potential factors of a **Web Based Grocery Shopping** system called iBuy, which is specified in the Scope & Requirements Document, and to use the OPM model they had created, shown in Figure 19 through Figure 22. The evaluation focused on the **Shopping List Creating** process, which is shown in Figure 22: SD1.3 - Shopping List Creating in-zoomed. There were two grocery shopping datasets with different difficulty levels that students were asked to handle: one of the Jerusalem shop and the other—of the Tel-Aviv shop.

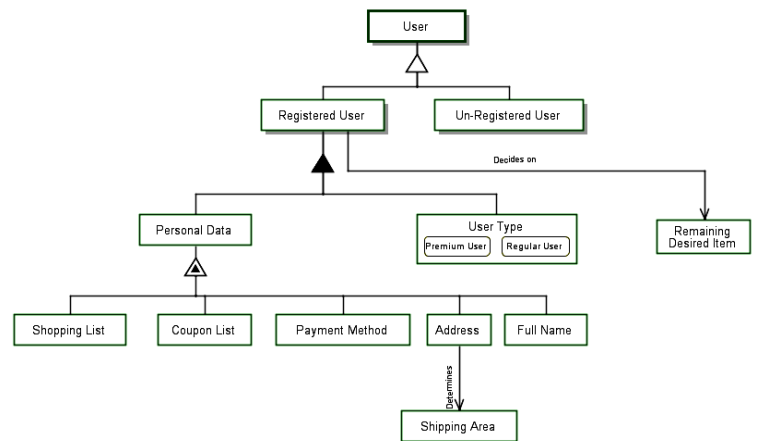


Figure 21: View 2 - **User** unfolded

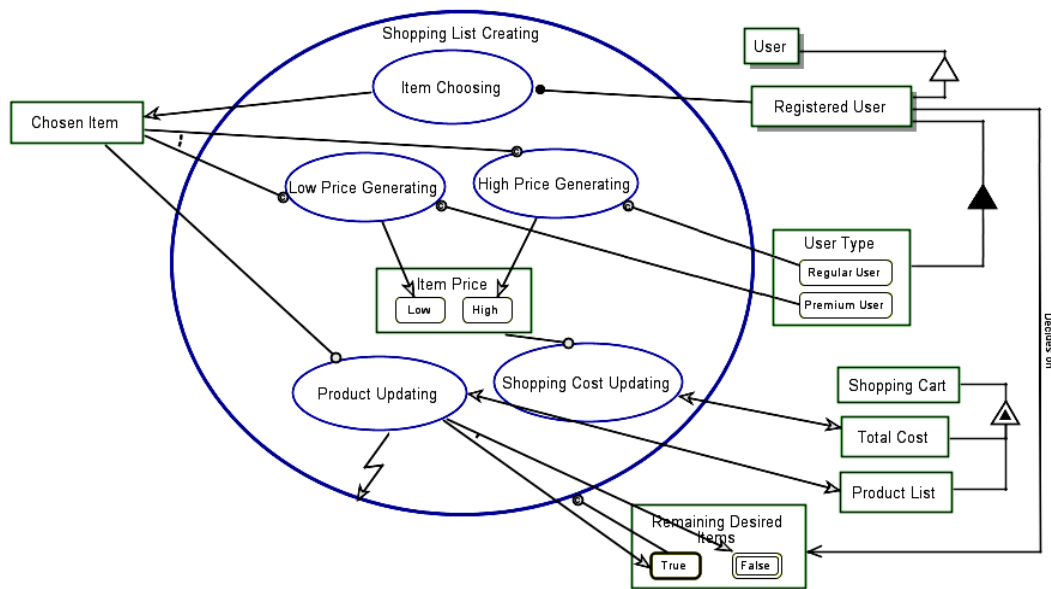


Figure 22: SD1.3 - Shopping List Creating in-zoomed

Our research hypothesis was that using OPM with the AUTOMATLAB approach would benefit the user in the following ways:

1. Users of AUTOMATLAB will gain deeper, more accurate understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB.
2. AUTOMATLAB users will understand the system's computational and quantitative aspects quicker than users who used OPM without AUTOMATLAB.
3. AUTOMATLAB users will be more confident in their understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB.
4. AUTOMATLAB users will understand the system's computational and quantitative aspects better, with less difficulty, than who used OPM without AUTOMATLAB.

The different factors we analyzed referred mainly to the **Shopping List Creating** process in the OPM model shown in Figure 21. The students were requested to evaluate the answer to four different questions, and answer a few questions regarding their evaluation, confidence, difficulty, and the time it took them to answer. The analysis was performed on two data sets with different levels of difficulty. The first data set was simpler in the sense that there were fewer customers, and the behavioral model was simpler. For example, in the simple data set, a customer starting as a regular user would either purchase his/her entire shopping list as a regular user, or pay a fee and become a premium user, whichever is cheaper for that purchase. In the advanced data set, when a regular buyer has the option to pay less if she or he becomes a premium user (buyer), the buyer's behavior is stochastic: s/he might choose

to stay a regular user, pay a fee and become a premium user, or cancel the entire purchase.

The students were asked to answer the following questions:

1. What type of customer is more profitable for the iBuy owner: Regular user or Premium user?
2. What are the three most profitable products for the iBuy owner?
3. What is the premium user monthly fee that will maximize the profit for the iBuy owner?
4. What is the premium user monthly fee that will make the amount of items purchased by regular users and premium users equal?

For each answer, the students were then asked to explain how they had deduced their answer, how accurate they thought their answer was and why, how difficult it was to complete their answer and why, and how long it took them to complete their answer. Since our sample was not sufficient for accurate statistical analysis, we combined qualitative analysis of the evaluation with the statistical analysis.

The students were given data sets containing a list of items sold by the **Web Based Grocery Shopping** system, monthly fee for premium users, a list of customers, and potential shopping lists for customers.

Identical questionnaires and data sets were given to both the experimental and control groups. The questionnaire included four main questions, such as "What are the three most profitable products for the iBuy owner?" Each one of the four questions was graded according to its correctness (*correct/incorrect*). In addition, students were asked to rate on a 1-5 Likert scale how accurate their answer is and how difficult for them was it to obtain this answer. Finally, students were asked to report about the time it took them to answer this question.

The control group was allowed to answer the questions using whatever tool they desire. The experimental group students received the automatically-generated MATLAB code from the AUTOMATLAB approach shown in Appendix B, Figure 23

through Figure 29, and were instructed to use it in order to answer the four questions. The MATLAB code was intended to help them gain better understanding of the model and to serve as a basis for a simulation of the system, including computational and stochastic aspects required to answer the four questions.

The students from the experimental group who received the AUTOMATLAB generated MATLAB code submitted their code when answering the questionnaire. An example of the **Shopping List Creating** enhanced code is presented in Appendix B, Figure 29.

A total of 96 answers from 24 questionnaires were graded according to their accuracy, and student explanations regarding difficulty, confidence in the outcome accuracy, and the time required to complete the assignment were analyzed qualitatively. The number of questionnaires submitted from each group is presented in Table 5.

Table 5: Amount of questionnaires submitted from each group

	Experimental Group		Control Group	
	Jerusalem Data set	Tel-Aviv Data set	Jerusalem Data set	Tel-Aviv Data set
Amount of questionnaires:	5	5	7	7

We analyze the data from the evaluation for the three variables: *group* (experimental or control), *level* ('Jerusalem' – level-1 – the easy data set, or 'Tel-Aviv' – level-2 – the difficult data set) and *question* (Q1, Q2, Q2, or Q4) using multi-way repeated measures tests with two within-subjects independent variables (*level*, *question*) and between-subjects independent variable (*group*). The dependent variable, namely *grade*, *time*, *confidence in answer accuracy*, and *difficulty* was changed in each hypothesis test. Independent t-test and one-way ANOVA with a Bonferroni correction served as our post-hoc tests, where it was needed.

#### First hypothesis analysis

Our first hypothesis was that users of AUTOMATLAB will gain deeper, more accurate understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB. In line with this hypothesis, we found a significant main effect for *group* ( $F(1, 10) = 5.23$ ,  $p < .05$ ,  $\eta^2 = .34$ ), indicating that students who used AUTOMATLAB scored higher ( $M = .71$ ,  $SD = .05$ ) than students who answered the questions without using AUTOMATLAB ( $M = .55$ ,  $SD = .05$ ). Likewise, there was a significant main effect for *level* ( $F(1,10) = 5.99$ ,  $p < .05$ ,  $\eta^2 = .37$ ), which means that the level-1 dataset yielded higher grades ( $M = .72$ ,  $SD = .06$ ) than level-2 ( $M = .54$ ,  $SD = .04$ ). Finally, we found a significant main effect for *question* ( $F(3, 30) = 6.15$ ,  $p < .01$ ,  $\eta^2 = .38$ ). Post-hoc analysis using Bonferroni correction revealed that Q4's grades ( $M = .29$ ,  $SD = .10$ ) were significantly lower than the grades of Q1 ( $M = .81$ ,  $SD = .07$ ) and Q2 ( $M = .82$ ,  $SD = .05$ ). The

grades of Q3 ( $M = .60$ ,  $SD = .12$ ) did not differ significantly from the rest of the questions. Significant interaction was found between *group* and *level* ( $F(1,10) = 4.88$ ,  $p < .05$ ,  $\eta^2 = .33$ ). In subsequent tests we found that this difference is due to the interaction between *group* and level-2 ( $F(1,10) = 16.77$ ,  $p < .01$ ,  $\eta^2 = .62$ ). The difference between the experimental and control groups is mainly due to Q2 ( $t(10) = 3.04$ ,  $p < .01$ ) and Q3 ( $t(10) = 1.86$ ,  $p < .05$ ), indicating that the grades of the experimental group on Q2 ( $M = 1$ ,  $SD = 0$ ) were higher than those of the control group ( $M = .55$ ,  $SD = .39$ ), and that the grades of the experimental group in Q3 ( $M = .80$ ,  $SD = .45$ ) were higher than the grades of the control group ( $M = .29$ ,  $SD = .49$ ).

#### Second hypothesis analysis

Our second hypothesis was that AUTOMATLAB users will understand the system's computational and quantitative aspects quicker than users who used OPM without AUTOMATLAB.

The main effect for *group* ( $F(1,10) = .39$ ,  $P > .05$ ,  $\eta^2 = .04$ ) was not significant, indicating that the experimental ( $M = 2.02$ ,  $SD = .53$ ) and control groups ( $M = 1.59$ ,  $SD = .45$ ) did not significantly differ in the time required to solve the questions. According to the hypothesis there was a significant main effect for *question* ( $F(3, 30) = 11.87$ ,  $p < .001$ ,  $\eta^2 = .54$ ). Post-hoc analysis using Bonferroni correction revealed that the time needed to achieve an answer for Q1 ( $M = 3.4$ ,  $SD = .56$ ) was significantly longer than the time needed for Q2 ( $M = 1.43$ ,  $SD = .37$ ), Q3 ( $M = 1.12$ ,  $SD = .39$ ) and Q4 ( $M = 1.29$ ,  $SD = .42$ ).

#### Third hypothesis analysis

Our third hypothesis was that AUTOMATLAB users will be more confident in their understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB. The main effect for *group* ( $F(1,10) = .62$ ,  $P > .05$ ,  $\eta^2 = .06$ ) was not significant, indicating that the experimental and control groups did not significantly differ in the confidence they had in the accuracy of their results. These test results indicate in addition that there was a significant main effect for *question* ( $F(3, 30) = 7.14$ ,  $p = .001$ ,  $\eta^2 = .42$ ). Post-hoc analysis using Bonferroni correction revealed that confidence in answers for Q1 ( $M = 4.41$ ,  $SD = .15$ ) were significantly higher than the grades for Q3 ( $M = 3.36$ ,  $SD = .22$ ) and Q4 ( $M = 3.70$ ,  $SD = .23$ ). Confidence in answers of Q2 ( $M = 3.96$ ,  $SD = .18$ ) did not differ significantly from the rest of the questions.

#### Fourth hypothesis analysis

Our fourth hypothesis was that AUTOMATLAB users will understand the system's computational and quantitative aspects better, with less difficulty, than who used OPM without AUTOMATLAB. The multi-way repeated measures



test revealed that the main effect for *group* ( $F(1, 10) = 4.00, p = .07, \eta^2 = .29$ ) has borderline significance. Since our hypothesis is one-tailed, we can deduct a significant difference between the groups, indicating that students who did not use AUTOMATLAB indicated a higher level of difficulty ( $M = 3.27, SD = .25$ ) than students who used AUTOMATLAB ( $M = 2.5, SD = .29$ ), suggesting that the experimental group subjectively experienced a lower level of difficulty than the control group when solving the questions.

The main effect for *level* ( $F(1,10) = .59, P > .05, \eta^2 = .06$ ) was not significant, but significant interaction was found between *group* and *level* ( $F(1,10) = 14.49, p < .005, \eta^2 = .59$ ). Follow-up tests revealed significant interaction between *group* and level-2 ( $t(10) = 3.09, p \leq .01$ ) resulting from difference between groups for Q2 in level-2 ( $t(10) = 2.71, p < .05$ ) and difference for Q4 in level-2 ( $t(10) = 2.72, p < .05$ ), as can be seen in Table 6.

Table 6: Results of Continued tests for interaction between *group* and *level*

	question	Experimental group		Control group		t(10)
		mean	std	mean	std	
	Q1	4.20	1.30	3.00	.82	1.972 <sup>a</sup>
	Q2	2.80	.84	3.29	1.11	.82
level-1	Q3	2.20	1.10	2.43	1.27	.32
	Q4	2.20	.84	3.43	.53	3.12*
	<b>Total</b>	<b>11.40</b>	<b>3.29</b>	<b>12.14</b>	<b>2.12</b>	<b>.48</b>
	Q1	2.60	1.34	3.71	1.11	1.57
	Q2	1.40	.55	3.14	1.35	2.71*
level-2	Q3	2.60	1.82	3.57	.98	1.21
	Q4	2.00	.71	3.57	1.13	2.72*
	<b>Total</b>	<b>8.60</b>	<b>3.78</b>	<b>14.0</b>	<b>2.31</b>	<b>3.09*</b>

\*  $P < .05$

<sup>a</sup> Borderline significance was found, for apposite of hypothesis.

Our first hypothesis was that users of AUTOMATLAB will gain deeper, more accurate understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB. The results indeed show that the experimental group achieved higher accuracy levels than the control group in our experiment. The experimental group increase in accuracy was more significant for the more complicated data set (Q2 and Q3), suggesting that the benefits of AUTOMATLAB are more prominent for more complex situations and needs.

Analysis of the explanations provided by the students when submitting their answers suggests that the students in the experimental group attempted to create a more accurate simulation of the system behavior. One of the subjects noted: *"The calculation is performed in the MATLAB code... For every customer I calculate the profit... Dealing with a customer is one iteration of the function 'ShopingListCreating'... The simulation results show..."*

Students in the control group used other method (mainly simple Excel spreadsheets or pen-and-paper calculations), ignoring seemingly unnecessary aspects: *"Didn't compare one by one..."*, *"Response was relatively difficult since lots of considerations came in and I needed to make assumptions"*.

This is assumed to be the cause for the difference in accuracy between their answers.

Our second hypothesis was that AUTOMATLAB users will understand the system's computational and quantitative aspects quicker than users who used OPM without AUTOMATLAB. The results did not show a conclusive difference between the experimental and control groups. We have seen that the time needed to answer Q1 was significantly longer than the time needed for Q2 through Q4. For the experimental group, we assume that the longer time required to answer the first question is due to the need to enhance the automatically generated MATLAB code when solving the first question, and this code was later used to solve the rest of the questions, as one student commented: *"Calculations were very similar to previous questions..."*, *"The code was already prepared – only one function needed to be changed"*.

Our third hypothesis was that AUTOMATLAB users will be more confident in their understanding of the system's computational and quantitative aspects than users who used OPM without AUTOMATLAB. The results received for this hypothesis were not conclusive. No clear differences were found for the different groups or datasets. Since the experimental group students had limited level of experience with MATLAB, perhaps some of their lack of confidence was due to the tool being used and not due to lack of understanding of the model. This may be a subject of a future research.

The fourth hypothesis was that AUTOMATLAB users will understand the system's computational and quantitative aspects better and with less difficulty than those who used OPM without AUTOMATLAB. The results indeed show that the students who did not use AUTOMATLAB rated their difficulty as higher.

Analysis of the students' explanations suggests that the difficulty in the control group is associated mainly with the challenge in representing the customer behavior model in simple tools like Excel or handwritten calculations: *"It took a long time to get the calculations since I didn't know the appropriate action in Excel."*, *"I had to go over every buyer and every product which is a lot of intersections!!!"*. Explanations of experimental group students for Q2-Q4 repeatedly mentioned using the previously created code as a reason for low difficulty, supporting our hypothesis: *"No changes were needed from previous..."*, *"From the way I implemented the solution of the answer in the first part... no more changes had to be made."*

From both the statistical and qualitative analyses of the evaluation we have seen that the AUTOMATLAB method does indeed benefit the user in several ways. AUTOMATLAB may improve the accuracy of understanding a system's quantitative aspects and decrease the difficulty of reaching such understanding. Results regarding the time needed to understand the quantitative aspects and user's confidence regarding her or his understanding are not significant, probably due to the lack of participants' experience with the

MATLAB environment. This aspect should be tested with more proficient participants in a future research.

## 11. CONCLUSION AND FUTURE RESEARCH

This research has tackled the problem of merging computational aspects and capabilities into conceptual models of systems, which are primarily qualitative in nature. Due to the level of abstraction of conceptual models, their computational capabilities are weak or missing altogether. Some modeling and simulation methods that do provide the computational aspects generally lack the high level of abstraction required from a conceptual modeling language. For example, Arena, a widely used discrete simulation software tool (Kelton et al., 2014) discussed above, or [ModelCenter](#), which integrates simulation tools from various vendors, can be used for this purpose, making it unnecessary to merge the computational aspects into the system. However, the abstraction power of OPM and its complexity management mechanisms make it highly suitable for early-stage conceptual modeling, so incorporating MATLAB's computational capabilities into OPM for seamless transition from conceptual to detailed design is highly desirable.

The computational simplification problem defined in this work relates to the difficulty of incorporating quantitative aspects into conceptual models, which focus primarily on key qualitative aspects of the system. We presented two possible solutions for this problem, based on expanding OPM to combine MATLAB and Simulink. In the first, AUTOMATLAB approach, we solve the computational simplification problem by adding a parallel MATLAB-based representation of the entire OPM model, which can be augmented with any desired computational aspects in the MATLAB representation. A major advantage of the AUTOMATLAB approach is that the OPM model becomes an integral part of the augmented MATLAB model, enabling it to evolve and serve for increasingly more quantitative-oriented simulations in downstream lifecycle stages of the system.

In the second, OPM/CS approach, we aim to solve the computational simplification problem by augmenting the capabilities of the OPM in-zooming mechanism. The additional capability provides for the content of an in-zoomed process to be replaced by a MATLAB function or a Simulink model containing the necessary computational aspects. The main advantages of the OPM/CS approach are (1) the simplicity of the enhancement that uses an intuitive extension of the OPM in-zooming mechanism, and (2) the preservation of the original OPM conceptual model with its semantics.

The two approaches were demonstrated via examples and case studies. An evaluation of the AUTOMATLAB approach was conducted with human subjects, showing benefits of this approach in terms of better system understanding. Results regarding user confidence in system understanding and time required to achieve such understanding were not conclusive, perhaps due to the small sample. The statistical results were supported by qualitative content analysis of the subjects' responses to questions.

Both approaches have been designed and implemented with forward compatibility to the future online OPM CASE tool, and partial compatibility to the development version of the current OPCAT. In a future OPM modeling tool, a large scale test and comparison of AUTOMATLAB and OPM/CS should be performed.

## REFERENCES

- [1] Dori, D. Object-Process Methodology – A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002.
- [2] Dori, D. Model-Based Systems Engineering with OPM and SysML. Springer, New York, 2015 (in press).
- [3] Somekh, J., Haimovich, G., Guterman, A., Dori, D., and Choder, M. Conceptual Modeling of mRNA Decay Provokes New Hypotheses. PLOS ONE, Sept. 2014. [PLOS ONE 9\(9\): e107085. doi:10.1371/journal.pone.0107085.](#)
- [4] Dori, D. Words from Pictures for Dual Channel Processing: A Bimodal Graphics-Text Representation of Complex Systems. Communications of the ACM, 51(5), pp. 47-52, 2008.
- [5] Dori, D., Linchevski, C., and Manor, R. OPCAT – A Software Environment for Object-Process Methodology Based Conceptual Modeling of Complex Systems. Proc. 1st International Conference on Modeling and Management of Engineering Processes, University of Cambridge, Cambridge, UK, Heisig, P., Clarkson, J., and Vajna, S. (Eds.), pp. 147-151, July 19-20, 2010.
- [6] Houcque, D. Introduction to MATLAB for Engineering Students. Northwestern University, Version 1.2, August 2005.
- [7] Mattsson, S., Elmqvist, H. Modelica – An International Effort to Design the Next Generation Modeling Language. 1997.
- [8] Harel, D., Marelly, R. Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-engine, Springer Verlag, 2003.
- [9] Schultz, M., Zerbe, V., and Marwedel, S. Using the Object Process Methodology to Build Simulation Models, Proc. 3rd International Conference on Model-Based Systems Engineering, Fairfax, VA, USA, 2010.
- [10] Mathworks. MATLAB & SYMULINK. Version 8.6, 2015. [http://www.mathworks.com/help/pdf\\_doc/simulink/sl\\_using.pdf](http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf)
- [11] Kelton D.W., Sadowski R.P., and Sadowski D.A. Simulation with Arena, 6th Edition. McGraw-Hill Professional, 2014.
- [12] Gilat, T. A Framework for Simulation of Discrete Events Systems Based on the Object-Process Methodology, Technion, PhD thesis, 2002.
- [13] Weikens, T. Systems Engineering with SysML/UML: Modeling, Analysis, Design, 2007.
- [14] Operational Semantics for OPM, Dov Dori, Ofer Strichman, Valeria Perelman. March 2011, DRAFT.
- [15] MathWorks Documentation Center MATLAB Functions, July 2011 <http://www.mathworks.com/help/releases/R2009b/helpdesk.html>
- [16] OMG Unified Modeling Language (OMG UML) Infrastructure, Version 2.4.1, Object Management Group, August 2011.
- [17] OMG Systems Modeling Language (OMG SysML), Version 1.3, Object Management Group, June 2012.
- [18] B.P. Zeigler. Theory of Modeling and Simulation. Wiley, New York. 1976
- [19] R. Davies, P. Roderick and J. Raftery. The Evaluation of Disease Prevention and Treatment using Simulation Models. European Journal of Operational Research, 150, 53-66. 2003.

- [20] R. Sinha, V.C. Liang, C.J.J. Paredis, and P.K. Khosla. Modeling and Simulation Methods for Design of Engineering Systems. *Journal of Computing and Information Science in Engineering*. Vol. 1, pp. 84-91, 2001.
- [21] S. Robinson. Conceptual Modeling for Simulation part I: Definition and Requirements. *Journal of the Operational Research Society*, Vol. 59, No. 3 (Mar., 2008), pp. 278-290
- [22] S. Robinson. *Conceptual Modeling for Simulation*. Wiley Encyclopedia of Operations Research and Management Science, 2010.
- [23] A. Maria. Introduction to Modeling and Simulation. *Proceedings of the 29th conference on winter simulation (WSC '97)*. 1997.
- [24] J. S. Carson, II. Introduction to Modeling and Simulation. *Proceedings of the 36th conference on winter simulation (WSC '04)*. 2004.
- [25] N. Sharma, and M.K. Gobbert, A comparative evaluation of MATLAB, Octave, FreeMat, and Scilab for research and teaching. 2010.
- [26] Scolnik, M. *Introduction to Radar Systems*. New York, NY:McGraw-Hill, 1980.
- [27] Peak, R.S., Fulton, R.E., Nishigaki, I., and Okamoto, N. Integrating Engineering Design and Analysis Using a Multi-Representation Approach. *Engineering with Computers* 14, pp. 93-114, 1998.
- [28] Blekhman, A., Dori, D. and Martin, R. Model-Based Standards Authoring. *Proc. 21st INCOSE International Symposium*, Denver, CO, USA, pp. 650-659, June 19-23, 2011.
- [29] Wölk, S., and Shea, K. A computational product model for conceptual design using SysML. *Proceedings of ASME IDETC/CIE*, San Diego, CA, USA, 2009.
- [30] Bergsjö, D., Almefelt, L., Dinar, M., and Malmqvist, J. Customizing Product Data Management for Systems Engineering in an Informal Lean-Influenced Organization. *Systems Research Forum*, 4(1), pp. 101-120, 2010.

APPENDIX A – OPM SUMMARY

1. Entities

Name	Symbol	OPL	Definitions
Things	Object 		An <b>object</b> is a thing that exists. A <b>process</b> is a thing that transforms at least one object. Transformation is object generation or consumption, or effect—a change in the state of an object.
	Process 	<b>B is physical.</b> <i>(shaded rectangle)</i>	
		<b>C is physical and environmental.</b> <i>(shaded dashed rectangle)</i>	
		<b>E is physical.</b> <i>(shaded ellipse)</i>	
		<b>F is physical and environmental.</b> <i>(shaded dashed ellipse)</i>	
State		<b>A is s1.</b> <b>B can be s1 or s2.</b> <b>C can be s1, s2, or s3.</b> <b>s1 is initial.</b> <b>s3 is final.</b>	A <b>state</b> is situation an object can be at or a value it can assume. States are always within the object that owns them. A state can be initial, final, or both.

2. Structural Links

Symbol	Name	OPL	Allowed Source-to-Destination connections
	Aggregation-Participation	<b>A consist of B.</b>	Object-Object Process- Process
	Exhibition-Characterization	<b>A exhibits B.</b>	Object-Object Object-Process Process-Object Process- Process
	Generalization-Specialization	<b>B is an A.</b> (objects) <b>B is A.</b> (processes)	Object-Object Process- Process
	Classification-Instantiation	<b>B is an instance of A.</b>	Object-Object Process- Process
	Tagged structural links: Unidirectional	According to text added by user	Object-Object
	Bidirectional		Process- Process

3. Fundamental Structural Links

Name	Symbol	OPL	Semantics
Aggregation-Participation 		<b>A consists of B and C.</b>	A is the whole, B and C are parts.
		<b>A consists of B and C.</b>	
Exhibition-Characterization 		<b>A exhibits B, as well as C.</b>	Object B is an attribute of A and process C is its operation (method). A can be an object or a process.
		<b>A exhibits B, as well as C.</b>	
Generalization-Specialization 		<b>B is an A.</b> <b>C is an A.</b>	A specializes into B and C. A, B, and C can be either all objects or all processes.
		<b>B is A.</b> <b>C is A.</b>	
Classification-Instantiation 		<b>B is an instance of A.</b> <b>C is an instance of A.</b>	Object A is the class, for which B and C are instances. Applicable to processes too.

4. Tagged Structural Links

Name	Symbol	OPL
Unidirectional & bidirectional tagged structural links		<b>A relates to B.</b> (for unidirectional) <b>A and C are related.</b> (for bidirectional)

5. Procedural Enabling Links

Name	Symbol	OPL	Semantics
Agent Link		A handles B.	Denotes that object A is a human operator who triggers process B.
Instrument Link		B requires A.	"Wait until" semantics: Process B cannot happen if object A does not exist.
State-Specified Instrument Link		B requires s1 A.	"Wait until" semantics: Process B cannot happen if object A is not at state s1.

6. Procedural Transforming Links

Name	Symbol	OPL	Semantics
Consumption Link		B consumes A.	Process B consumes Object A.
State-Specified Consumption Link		B consumes s1 A.	Process B consumes Object A when it is at State s1.
Result Link		B yields A.	Process B creates Object A.
State-Specified Result Link		B yields s1 A.	Process B creates Object A at State s1.
Effect Link		B affects A.	Process B changes the state of Object A; the details of the effect may be added at a lower level.
State-Specified Effect Link (Input-Output Links Pair)		B changes A from s1 to s2.	Process B changes the state of Object A from State s1 to State s2.

7. Procedural Links: Control Links

Name	Symbol	OPL	Semantics
Instrument Event Link		A triggers B. B requires A.	Generation of object A is an event that triggers process B. B will start executing if its precondition is met. Since A is

			instrument it will not be affected by B.
State-Specified Instrument Event Link		A triggers B when it enters s1. B requires s1 A.	Entering state s1 of object A is an event that triggers process B. B will start executing if its precondition is met. Since A is instrument it will not be affected by B.
Consumption Event Link		A triggers B. B consumes A.	Generation of object A is an event that triggers process B. B will start executing if its precondition is met, and if so it will consume A.
State-Specified Consumption Event Link		A triggers B when it enters s2. B consumes s2 A.	Entering state s2 of A is an event that triggers process B. If B is triggered, it will consume A. B will start executing if its precondition is met, and if so it will consume A.
Condition Link		B occurs if A exists.	Existence of object A is a condition for the execution of B. If A does not exist, then B is skipped and regular system flow continues.
State-Specified Condition Link		B occurs if A is s1.	Existence of object A at state s2 is a condition for the execution of B. If A is not in s2, then B is skipped and regular system flow continues.
Invocation Link		B invokes C.	Execution termination of process B is an event that triggers process C. B yields a temporary object that is immediately consumed by C and therefore not be shown explicitly in the model.
Exception Link		A triggers B when it lasts more than 4 seconds.	Process A has to be assigned with maximal acceptable time duration, which, if exceeded, triggers process B.

## Appendix B. MATLAB code generated automatically from the OPM processes of the iBuy System

```

1  function [TotalCost,ProductList,RemainingDesiredItems,ChosenItem] =...
2  ShoppingListCreating(RegisteredUser,UserEntity,TotalCost,ProductList,RemainingDesiredItems)
3  % ShoppingListCreating zooms into ItemChoosing, LowPriceGenerating, HighPriceGenerating,...
4  % ShoppingCostUpdating, and ProductUpdating, as well as Item Price.
5
6  while isequal(RemainingDesiredItems,'True')
7  % ShoppingListCreating occurs if RemainingDesiredItems is True.
8
9  if ~isempty(RegisteredUser)
10 % RegisteredUser handles ItemChoosing.
11 [ChosenItem] = ItemChoosing(RegisteredUser);
12 % ShoppingListCreating consists of ItemChoosing
13 % ItemChoosing yields ChosenItem.
14 end
15
16 if ~isempty(ChosenItem) && isequal(UserEntity,'Regular user')
17 % HighPriceGenerating occurs if ChosenItem is in existent and UserEntity is Regular user.
18 [ItemPrice] = HighPriceGenerating(ChosenItem,UserEntity);
19 % ShoppingListCreating consists of HighPriceGenerating
20 % HighPriceGenerating yields High ItemPrice.
21 end
22
23 if ~isempty(ChosenItem) && isequal(UserEntity,'Premium user')
24 % LowPriceGenerating occurs if ChosenItem is in existent and UserEntity is Premium user.
25 [ItemPrice] = LowPriceGenerating(ChosenItem,UserEntity);
26 % ShoppingListCreating consists of LowPriceGenerating
27 % LowPriceGenerating yields Low ItemPrice.
28 end
29
30 if ~isempty(ItemPrice) && ~isempty(TotalCost)
31 % ShoppingCostUpdating requires ItemPrice.
32 [TotalCost] = ShoppingCostUpdating(ItemPrice,TotalCost);
33 % ShoppingListCreating consists of ShoppingCostUpdating
34 % ShoppingCostUpdating affects TotalCost.
35 end
36
37 if ~isempty(ChosenItem) && ~isempty(ProductList)
38 % ProductUpdating requires ChosenItem.
39 [RemainingDesiredItems,ProductList] = ProductUpdating(ChosenItem,ProductList);
40 % ShoppingListCreating consists of ProductUpdating
41 % ProductUpdating affects ProductList.
42 % ProductUpdating yields either True RemainingDesiredItems or False RemainingDesiredItems.
43 end
44
45 % ProductUpdating invokes ShoppingListCreating.
46 end
47
48 end

```

Figure 23: MATLAB code generated from the process **Shopping List Creating**

```

1  function [ChosenItem] = ItemChoosing(RegisteredUser)
2  % RegisteredUser handles ItemChoosing.
3  % ItemChoosing yields ChosenItem.
4
5
6  % [] = RegisteredUser; % RegisteredUser handles ItemChoosing.
7
8  ChosenItem = 1; % ItemChoosing yields ChosenItem.
9
10 end

```

Figure 24: MATLAB code generated from the process **Item Choosing**

```

1  function [ItemPrice] = HighPriceGenerating(ChosenItem,UserEntity)
2  % HighPriceGenerating occurs if ChosenItem is in existent and UserEntity is Regular user.
3  % HighPriceGenerating yields High ItemPrice.
4
5  -
6
7  % [] = ChosenItem; % HighPriceGenerating occurs if ChosenItem is in existent
8
9  % [] = UserEntity; % HighPriceGenerating occurs if UserEntity is Regular user
10
11 - ItemPrice = 'High'; % HighPriceGenerating yields High ItemPrice.
12
13 - end

```

Figure 25: MATLAB code generated from the process **High Price Generating**

```

1  function [ItemPrice] = LowPriceGenerating(ChosenItem,UserEntity)
2  % LowPriceGenerating occurs if ChosenItem is in existent and UserEntity is Premium user.
3  % LowPriceGenerating yields Low ItemPrice.
4
5
6
7  % [] = ChosenItem; % LowPriceGenerating occurs if ChosenItem is in existent
8
9  % [] = UserEntity; % LowPriceGenerating occurs if UserEntity is Premium user
10
11 - ItemPrice = 'Low'; % LowPriceGenerating yields Low ItemPrice.
12
13 - end

```

Figure 26: MATLAB code generated from the process **Low Price Generating**

```

1  function [TotalCost] = ShoppingCostUpdating(ItemPrice,TotalCost)
2  % ShoppingCostUpdating requires ItemPrice.
3  % ShoppingCostUpdating affects TotalCost.
4
5  -
6
7  % [] = ItemPrice; % ShoppingCostUpdating requires ItemPrice.
8
9  [TotalCost] = TotalCost; % ShoppingCostUpdating affects TotalCost.
10 - end

```

Figure 27: MATLAB code generated from the process **Shopping Cost Updating**

```

1  function [RemainingDesiredItems,ProductList] = ProductUpdating(ChosenItem,ProductList)
2  % ProductUpdating requires ChosenItem.
3  % ProductUpdating affects ProductList.
4  % ProductUpdating yields either True RemainingDesiredItems or False RemainingDesiredItems.
5
6
7  % [] = ProductUpdating; % ShoppingCostUpdating requires ProductUpdating.
8
9  [ProductList] = ProductList; % ProductUpdating affects ProductList.
10
11
12 - RemainingDesiredItems = 'True';
13 % ProductUpdating yields either True RemainingDesiredItems or False RemainingDesiredItems.
14 RemainingDesiredItems = 'False';
15 % ProductUpdating yields either True RemainingDesiredItems or False RemainingDesiredItems.
16
17 - end

```

Figure 28: MATLAB code generated from the process **Product Updating**

```

1  function [TotalCost,TotalCostForPremuim,TotalRetailerCost,ItemsList,...
2      TotalPurchasedProducts] = ShoppingListCreating(RegisteredUser,UserEntity,...
3      ItemsList, IsNewPremiumUser)
4
5  % ProductList = zeros(length(RegisteredUser),2);
6  % ProductListIndex = 0;
7  RemainingDesiredItems = 'True';
8  itemIndex = 0;
9  TotalCost = 0;
10 TotalCostForPremuim = 0;
11 TotalRetailerCost = 0;
12 TotalPurchasedProducts = 0;
13
14 while isequal(RemainingDesiredItems,'True')
15     % ShoppingListCreating occurs if RemainingDesiredItems is True.
16     itemIndex = itemIndex+1;
17
18     if ~isempty(RegisteredUser)
19         [ChosenItem] = ItemChoosing(RegisteredUser, itemIndex);
20     end
21
22     [ItemPrice, ItemPriceWithDiscount,itemRetailerCost,ItemIndexInList] =...
23     PriceGenerating(ChosenItem,ItemsList);
24
25     if ~isempty(ChosenItem) && isequal(UserEntity,'Regular user')
26         %update amount of buyed products
27         TotalPurchasedProducts = TotalPurchasedProducts + ChosenItem(2);
28         [ItemsList] = UpdateTotalBuyedProducts( ItemsList, ItemIndexInList,...
29             ChosenItem(2), UserEntity, IsNewPremiumUser, 1 );
30     end
31
32     if ~isempty(ChosenItem) && isequal(UserEntity,'Premium user')
33         IsBuying = IsBuyingProduct(ItemPrice, ItemPriceWithDiscount );
34         if (IsBuying == 1)
35             TotalPurchasedProducts = TotalPurchasedProducts + ChosenItem(2);
36             ItemPrice = ItemPriceWithDiscount;
37         else
38             ItemPrice = 0;
39             itemRetailerCost = 0;
40         end
41         [ItemsList] = UpdateTotalBuyedProducts( ItemsList, ItemIndexInList,...
42             ChosenItem(2), UserEntity, IsNewPremiumUser, IsBuying );
43     end
44
45     if ~isempty(ItemPrice) && ~isempty(TotalCost)
46         [TotalCost] = ShoppingCostUpdating(ItemPrice,TotalCost);
47         [TotalCostForPremuim] = ShoppingCostUpdating(ItemPriceWithDiscount,...
48             TotalCostForPremuim); %low price for ALL the shopping list
49         TotalRetailerCost = TotalRetailerCost + itemRetailerCost;
50     end
51
52     % ProductUpdating invokes ShoppingListCreating.
53     if (itemIndex>= length(RegisteredUser))
54         RemainingDesiredItems = 'False';
55     end
56 end
57 end
58

```

Figure 29: Shopping List Creating enhanced code from AUTOMATLAB evaluation