# Model-Based Systems Engineering with OPM and SysML

Dov Dori

# Model-Based Systems Engineering with OPM and SysML

Foreword by Edward Crawley

Springer

Dov Dori
Technion, Israel Institute of Technology
Haifa, Israel

Massachusetts Institute of Technology
Cambridge, MA, USA

Chapter slides and end-of-chapter Q&As can be found at http://esml.iem.technion.ac.il/qanswer/

ISBN 978-1-4939-3294-8 ISBN 978-1-4939-3295-5 (eBook)
DOI 10.1007/978-1-4939-3295-5

Library of Congress Control Number: 2015954175

Springer New York Heidelberg Dordrecht London

Printed on acid-free paper

Springer Science+Business Media LLC New York is part of Springer Science+Business Media (www.springer.com)

# Foreword

Architecting and engineering large, complex socio-technical systems, as well as gaining deep understanding of existing natural and man-made systems, have eluded people for many years. Our thinking about systems and their role in improving humans' quality of life has evolved over the last two decades. We now understand better that successful systems do not materialize in a haphazard way. Rather, they must be carefully architected just like edifices, accounting for the needs, wants and requirements of their intended beneficiaries, and alternative architectures—ways in which these desired functions are embodied in form. These early decisions are critical to the system-to-be, as they determine the concept to be followed and consequently the whole direction the system development takes and the nature of the final outcome: how well the system performs in terms of delivering value, i.e., benefit at cost, while maintaining the other requirements of safety, robustness, ease of use, environmental friendliness, and many others.

As I was gaining these insights some 15 years ago, I realized that no matter how convincing your ideas are, and how compelling are the arguments, there is only so much one can do with preaching and hand-waving. It became obvious to me that making progress in the area of architecting and engineering complex systems is contingent upon a solid foundation of language and methodology. It so happened, that at that time, around year 2000, Dov stepped into my office, the office of the Head of the Aero-Astro Department at MIT, with a draft of his first book, titled *Object-Process Methodology—A Holistic Systems Paradigm* (Springer, 2002). Reading this draft in a plane I immediately understood that what I was holding in my hands was exactly what I was looking for.

Object-Process Methodology (OPM) is a systems modeling paradigm that represents the two things inherent in a system: its objects and processes. OPM is fundamentally simple; it builds on a minimal set of concepts: stateful objects—things that exist, and processes—things that happen and transform objects by creating or consuming them or by changing their states. This duality is recognized throughout the community who studies systems, and sometimes goes by labels such as form/function, structure/function, and functional requirements/design parameters. Objects are what a system or product is. Processes are what a system does. Yet, it is remarkable that so few modeling frameworks explicitly recognize this duality. As a result, designers and engineers try to jump from the goals of a system (the requirements or the "program") immediately to the objects. Serious theory in such disparate disciplines as software design, mechanical design and civil architectural design all recognize the value of thinking about processes in parallel with objects. Not only does OPM represent both objects and processes, but it explicitly shows the connections between them.

OPM has another fundamental advantage—it represents the system simultaneously in formal graphics and natural language. The two are completely interchangeable, conveying the exact same information. The advantage in this approach lies in appreciating the human limitation to the understanding of complexity. As systems become more complex, the primary barrier to success is the ability of the human designers and analysts to understand the complexity of the interrelationships. By representing the system in both textual and graphical form, the power of "both sides of the brain"—the visual interpreter and the language interpreter—is engaged. These are two of the strongest processing capabilities that are hard-wired into the human brain. Since each model fact is expressed both graphically and textually, in a subset of natural English, it is readily accessible to non-technical stakeholders, enabling them to take part in the early, critical stages of the system architecting and development, in which the most important decisions are made.

OPM allows a clear representation of the many important features of a system: its topological connections; its decomposition into elements and sub-elements; the interfaces among elements; and the emergence of function from elements. The builder of viewer of the model can view abstractions, or zoom into detail. One can see how specification migrates to implementation. These various views are invaluable when pondering the complexity of a real modern product system.

OPM semantics was originally geared towards systems engineering, as it can model information, hardware, people, and regulation. However, in recent years OPM started to serve also researchers in molecular biology, yielding tangible published new findings related to the mRNA lifecycle. This is a clear indication of the universality of the object and process ontology: As it turns out, one can use this minimal set of concepts to model systems in virtually any domain. Perhaps one exception is quantum physics, where our macro notions of particle (object?) and wave (process?), as well as matter (object?) and energy (process?) get fuzzy as we try to 'inspect' subatomic particles such as electrons. For any system from the molecular level up, all the way to the most complex natural, socio-technical and societal systems, the object-process paradigm works extremely well. OPM models concurrently explicate the function (utility), structure (form) and behavior (dynamics) of systems in a single, coherent model that uses one kind of diagram at any desired number of levels of detail by drilling down into the details of processes hand-in-hand with the objects they transform. The set of these self-similar object-process diagrams are represented not just visually, but also textually, catering to humans' dual channel processing, a key cognitive assumption of how we process information to convert it into actionable knowledge.

Having realized the value of OPM to systems architecting and engineering, I adopted it in my thinking and teaching, and it has become an important cornerstone of courses I have been teaching in systems architecture at MIT and elsewhere. In particular, OPM has become a key element in the teaching of core courses in the Systems Design and Management graduate program at MIT. I have used OPM in the SDM System Architecture course. It has proved an invaluable tool to professional learners in developing models of complex technical systems, such as automobiles, spacecraft and software systems. It allows an explicit representation of the form/function duality, and provides an environment in which various architectural options can be examined. The addition of OPM to my subject has added the degree of rigor of analysis necessary to move the study of technical system architecture towards that of an engineering discipline.

OPM is also used as a representational framework in the new book which I co-authored, *System Architecture: Strategy and Product Development for Complex Systems* (Pearson, 2015), which develops an approach to architecture and demonstrates it with examples ranging from pumps, circuits, and sorting algorithms, to complex systems in networking and hybrid cars. Indeed, the task of architecting and engineering a new system has become more complicated by the increasing number of components involved, the number of disparate disciplines needed to undertake the task, and the growing size of the organizations involved. Despite the common experience that members of many organizations share, they often lack a common product development vocabulary or modeling framework. Such a framework should be based on system science, be able to represent all the important interactions in a system, and be broadly applicable to electrical, informational, mechanical, optical, thermal, and human components.

OPM provides such a framework. Indeed, in 2008, the task force of the International Council on Systems Engineering (INCOSE) has recognized OPM as one of the six leading model-based systems engineering methodologies. Looking at the historical development of engineering disciplines, it is an appropriate time for such a rigorous framework to emerge. Disciplines often move through a progressive maturation. Early in the history of an intellectual discipline, we find observation of nature or practice, which quickly evolves through a period in which things are classified. A breakthrough often occurs when classified observations are abstracted and quantified. These phases characterize much of the work done to date in systems engineering and product development. Mature disciplines, such as mechanics, are well into the era of symbolic manipulation and prediction. Maturing disciplines such as human genomics are in the phase of symbolic representation.

OPM is a parallel development in symbolic representation of systems. Over the past two decades, the understanding of the need for systematic modeling capability has broadened. As OPM was developed in response to this growing recognition, so have other approaches. The most notable of these are UML, which includes 13 kinds of diagrams, geared for software engineering, and its derivative, SysML, which includes nine kinds of diagrams, designed more generally for systems engineering. Both SysML and OPM are listed as leading standards in the Guide to the Systems Engineering Body of Knowledge (SEBoK), an online ongoing project jointly sponsored by the Systems Engineering Research Center (SERC), the International Council on Systems Engineering (INCOSE), and the Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS).

SysML and OPM represent two different approaches to system modeling. In SysML up to nine diagrams are used, which are independently derived, and may not be completely consistent. In OPM one main diagram emerges. The need to integrate several kinds of diagrams may be more *complicated*. I make the distinction between *complexity*—the inherent fact that a system contains many parts interacting in multiple, often inexplicable ways, and *complicatedness*—the way a system model is presented through a certain modeling language and is perceived by a user. While there is not much we can do to reduce systems' inherent complexities, we can and should strive to reduce the complicatedness of the representation to the bare necessities without sacrificing accuracy and details. OPM with its minimal ontology of stateful objects and processes favorably responds to this challenge.

While the emphasis of the book is on OPM, because of the relatively wider spread use of SysML, Dov has included SysML with adequate presentation of its syntax and semantics, as well as synergies with OPM, comparison with OPM in terms of such factors as length of the standard specification, and ways OPM can replace many SysML diagram kinds with a single diagram kind. The coverage of both languages in the same book is unique, as Model-Based System Engineering (MBSE) books to date have mostly used SysML. This dual coverage of OPM and SysML is highly valuable, since the reader gets deeper perspective on MBSE that penetrates beneath the idiosyncrasies of a particular conceptual modeling language.

I recommend using this textbook for an intermediate or advanced course in model-based system engineering, product development, engineering design, and software engineering. It would be ideal for courses that attempt to show how various disciplines come together to form a multi-disciplinary product. With OPM now formally recognized as an ISO specification, it can form the backbone of a corporate or enterprise modeling framework for technical products and large-scale socio-technical systems. Such a representation would be especially valuable in conceptual and preliminary design, when much of the value, cost and risk of a product are established, but when few other modeling frameworks are available for decision support.[1]

*Professor Edward Crawley* *Massachusetts Institute of Technology, July 2015*

[1]Edward Crawley is the Ford Professor of Engineering and a Professor of Aeronautics and Astronautics at MIT. He is a member of the U.S. National Academy of Engineering and serves as the President of the Skolkovo Institute of Science and Technology in Moscow. Prof. Crawley is the first author of two recent books: "System Architecture: Strategy and Product Development for Complex Systems" and "Rethinking Engineering Education, the CDIO Approach".

# Preface

The quest for simplicity in a complex world has occupied thinkers for millennia. How to conceptualize what humans observe around them and what they wish to design in order to improve the quality of people's lives has been one of the major driving forces in advancing civilization. The advent of computers in the middle of the previous century was a great impetus to fostering thoughts about how to conceptually represent things in the real world. The initial accepted train of thought produced procedural programming, which put procedures, routines, functions, etc. at the center of programming. Further contemplations have led to the idea of putting objects, which are more static in nature, as the anchor of programs. The shift to the object oriented (OO) paradigm for programming languages, which occurred in the 1980s and 1990s, was followed by the idea that programming should be preceded by analysis and design of the programs, or, more generally, the systems those programs represent and serve. Naturally, the approach which was taken is also object-oriented.

In the early 1990, a plethora of some three dozen object-oriented analysis and design methods and notations flourished, leading to what was known as the "Methods War". Around that time, in 1991, when I moved from University of Kansas to Technion, Israel Institute of Technology, as I was tasked with teaching software design, I got interested in these topics. It was not long before I realized that just as the procedural approach to software was inadequate, so was the "pure" OO approach, which puts objects as the sole "first class" citizens, with "methods" (or "services") being their second-class subordinate procedures. However, I could not put my finger on what was missing.

My Eureka moment was in 1993, when I and colleagues from University of Washington were trying to model a system for automated transforming of hand-made engineering drawings to CAD models, a topic around which my research focused during that time. Drawing objects as the model's building blocks and connecting them on the white board, it dawned on me that not all the boxes in the model were really objects; some were things that *happen* to objects. When I circled those things, a pattern of a bipartite graph emerged, where the nodes representing objects—the things that exist—were mediated by those circled nodes, which I immediately called processes. This was the first object-process diagram (OPD) ever drawn. I realized then that the pendulum of the previously accepted procedural software to the primarily static OO paradigm moved too drastically. While the shift from procedures to objects as the focus of interest was a right move, it went too far, as it suppressed the systems' procedural aspect, which is essential to faithfully describe how systems change over time.

Forbidding processes, such as cake baking or check cashing, from being conceptual entities in their own right, and allowing their representation only as methods of object classes, results in distorted models, in which a check "owns" the cashing method or the cake owns the baking process. In real life, however, baking is a pattern of transformation of ingredients making up the dough that requires a baker, an oven, and energy to prepare the dough and convert it into a cake. Similarly, a check cannot cash itself; it requires a check writer having an account with sufficient funds, a check casher, and a bank clerk or an ATM. Each of the objects involved in these methods could just as well be the owner of the method. Modeling baking and cashing as stand-alone processes—conceptual things that represent physical or informatical object transformation patterns—open the door for creating models that are much more faithful to the way we conceive reality and convey it to others.

Indeed, recognizing processes as bona fide conceptual modeling building blocks beside, rather than underneath objects, is the prime foundation of Object-Process Methodology (OPM). OPM is founded on a universal minimal ontology, according to which objects exist, while processes transform them. Transformation includes object creation and consumption, as well as change of the state of an object. Therefore, OPM objects are stateful—they can have states. Hence, stateful objects and processes that transform them are the only two concepts in OPM's universal minimal ontology. Two other cornerstones of OPM are its bimodal graphical-textual representation and its built- in refinement-abstraction complexity management mechanisms of in-zooming and unfolding of a single type of diagram—OPD.

When I tried to publish a paper titled "Object-Process Analysis: Maintaining the Balance between System Structure and Behavior" with the buds of these ideas in 1993, it was serially rejected off hand with claims along the line that it had already been proven that what I was suggesting is impossible, like "mixing water with oil." Finally, the *Journal of Logic and Computation* accepted it, perhaps because being mathematics- rather than software-oriented, it was more tolerant toward ideas that went against the then new and glorious OO paradigm.

Meanwhile, in 1997, the "Methods Wars" culminated in the adoption of the Unified Modeling Languages (UML), by the Object Management Group (OMG), making it the de-facto standard for software design. UML 1 had nine types of diagrams. In 2000, when I attended a Technical Meeting of OMG in which UML was considered for progression from version 1 to 2, I proposed considering UML for being extended to handle not just software systems, but systems at large, a proposal that was dismissed off-hand by most attendees, who were software people. However, following a 2001 initiative of the International Council on Systems Engineering (INCOSE), in 2003 OMG issued the UML for Systems Engineering Request for Proposals, and in 2006 OMG adopted SysML (Systems Modeling Language) 1.0 specification, which is based on UML 2. Since then, SysML has become the de-facto standard for systems engineering.

Meanwhile, the first book on OPM, *Object-Process Methodology—a Holistic Systems Paradigm*, (Dori, 2002) was published, and OPM has been successfully applied and papers published in many diverse domains, ranging from the Semantic Web to defense and to molecular biology. In December 2015, after six years of work, ISO adopted and published OPM as ISO 19450—*Automation systems and integration—Object-Process Methodology*.

The realization and recognition that models can and should become the central artifact in system lifecycles has been gaining momentum in recent years, giving rise to model-based systems engineering (MBSE) as an evolving filed in the area of systems engineering. SysML and OPM have been serving as the two MBSE languages, but since SysML was adopted as a standard about eight years before OPM and has been backed by top-notch vendors, its adoption is currently more widespread. However, OPM is rapidly gaining acceptance in academia and its application in diverse industry segments is spreading.

This textbook, designed for both self-learning and as an undergraduate or graduate course, endows its readers with deep understanding of MBSE ideas, principles, and applications through modeling systems using both OPM and SysML. The book is comprised of three parts that encompass 24 chapters. Each chapter ends with a bulleted summary and a set of problems. Solutions to problems may be available in http://esml.iem.technion.ac.il/.

Part I introduces OPM and SysML via step-by-step modeling of a car automatic crash response system. Chapter 1 starts with a description of the system and its initial OPM model. In Chap 2 we enhance the model with text and animated simulation. Chapter 3 introduces links that connect things in the model. In Chap. 4 we introduce and use SysML's first three diagrams. Chapter 5 presents ways for managing the complexity of systems, while the dynamic aspect of the system is modeled in Chaps. 6 and 7. Abstraction and refinement mechanisms as means to manage complexity are the focus of Chap. 8, the last chapter in Part I.

Part II, *Model-Based Systems Engineering Fundamentals*, is a formal, theory-grounded exposure to OPM and SysML that discusses MBSE ontology, conceptual modeling constructs, and applications. Chapter 9 introduces and defines conceptual modeling. Chapter 10 presents the two basic building blocks of OPM—objects and processes, while Chap. 11 is about the textual modality of OPM—OPL. In Chap. 12 we turn to an orderly study of SysML with its four pillars and nine kinds of diagrams. The dynamic, time-dependent aspect of systems is the focus of Chap. 13, followed by studying the structural, time-independent system aspect in Chap. 14. Following Chap. 15, which deals with participation constraints and fork links, in Chap. 16 we introduce the four fundamental structural relations.

In Part III, *Structure and Behavior: Diving In*, we go to the heart of conceptual modeling, elaborating on the four fundamental structural relations and whole system aspects, including complexity management and control. Chapters 17 and 18 discuss aggregation-participation and exhibition-characterization, respectively. Chapter 19 is about states and values, concepts that are needed for generalization-specialization and classification-instantiation, both of which are elaborated on in Chap. 20. Chapter 21 concerns complexity management and the refinement-abstraction mechanisms of OPM, as well as complexity management in SysML. Chapter 22 is about OPM operational semantics and control links—the way control is managed during execution of the system. In Chap. 23 we specify how to model logical operators and probabilities. Finally, Chap. 24 is an overview of ISO 19450—*Automation Systems and Integration—Object-Process Methodology*, adopted by the International organization for Standardization in December 2015.

With respect to OPM, this book can be considered a superset of ISO 19450. While OPM, as specified in this book, is ISO 19450-complaint, the book provides in-depth motivation, rationale, and philosophical foundations for decisions made during the design of ISO 19450. These cannot be elaborated on in a standard, which, by its nature, is expected to be short and decisive, with little justifications. OPM points in the book that are not covered in ISO 19450 can be considered optional, or, in ISO nomenclature, *informative*, as opposed to *normative*—abiding ISO specifications.

This book is a product of six years of work, during which I have made all efforts to make it accurate, consistent, and formal, while also not lose the human touch and the interest of the future reader. It is my sincere hope that the book will serve as a reliable reference to MBSE in general and to OPM and SysML in particular.

Examining the above word cloud of this book (created by a program developed skillfully by Jason Davies),[2] based on close to 140,000 words contained in this book, we can see that the most frequent words are *process*, *object*, and *link*. Indeed, this is a most faithful testimony that OPM focuses on how to *model systems* (two other most frequent words in the cloud) by relating *processes* to *objects* using *links*. *Relation* is there too, along with other notable words, including *diagram*, *attribute*, *structural*, *procedural*, *semantics*, *state*, *control*, *change*, *effect*, *agent*, *time*, *constraint*, and *function*. Of course, *SysML* is there between *process* and *model*, near *OPD* (Object-Process Diagram—OPM's graphical modality) and *OPL* (Object-Process Language—OPM's textual modality). This list gives a good idea of what this book is about.

I wish to thank my three MIT collaborators, Prof. Ed Crawley and Prof. Oli de Weck from Engineering Systems Division and the Aero-Astro Department, and Pat Hale, Director of Systems Design and Management Program. Special thanks to my PhD student, Yaniv Mordecai, who provided insightful comments on many of the chapters in this book. I thank the Technion, Israel Institute of Technology, which provided me with the environment to develop OPM and with the 2013-4 sabbatical to complete this book. Finally, I wish to thank my beloved wife, Prof. Judy Dori, who provided pedagogical guidance and moral support, which made it possible for me to finish the book.

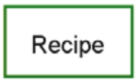
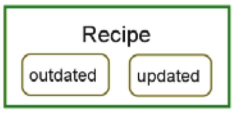
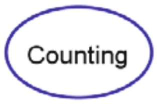
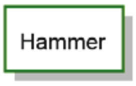
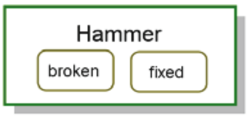
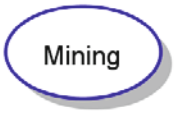
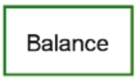
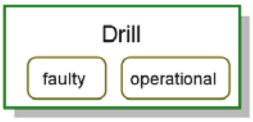
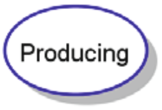
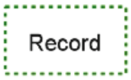
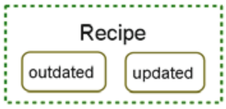
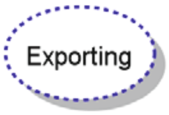
*Dov Dori* *Massachusetts Institute of Technology, July 2015*

[2]https://www.jasondavies.com/wordcloud/

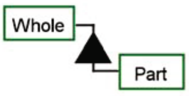
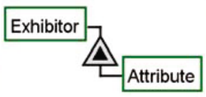
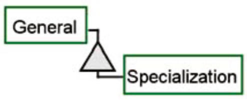
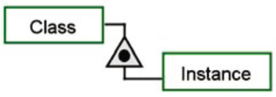
# Table of Contents

## Main ISO 19450-compliant OPM Symbols

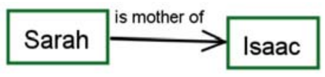
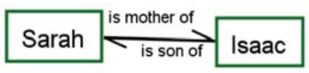
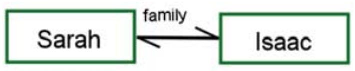
Things: stateful objects and processes

| thing property | value (notation) | thing | | |
|---|---|---|---|---|
| | | object | stateful object | process |
| essence | Informatical (flat) |  |  |  |
| | Physical (shaded) |  |  |  |
| affiliation | Systemic (solid) |  |  |  |
| | Environmental (dashed) |  |  |  |

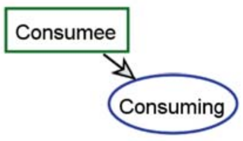
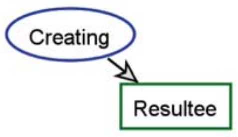
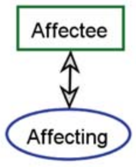
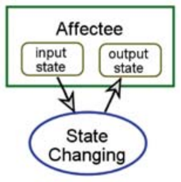
Fundamental structural links

| modality | aggregation-participation | exhibition-characterization | generalization-specialization | classification-instantiation |
|---|---|---|---|---|
| Graphics – Object-Process Diagram (OPD) |  |  |  |  |
| Textual – Object-Process Language (OPL) | **Whole** consists of **Part.** | **Exhibitor** exhibits **Attribute.** | **Specialization** is a **General.** | **Instance** is an instance of **Class.** |

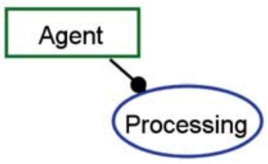
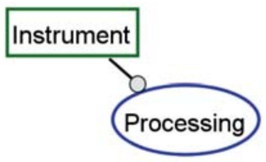
Tagged structural links

| unidirectional tagged link | bidirectional tagged link | Reciprocal tagged link |
|---|---|---|
|  **Sarah is mother of Isaac.** |  **Sarah is mother of Isaac. Isaac is son of Sarah.** |  **Sarah** and **Isaac** are **family.** |

Procedural transforming links

| consumption link | result link | effect link | in-out link pair |
|---|---|---|---|
|  **Consuming** consumes **Consumee.** |  **Creating** yields **Resultee.** |  **Affecting** affects **Affectee.** |  **State Changing** changes **Affectee** from **input state** to **output state.** |

Procedural enabling links

| agent link | instrument link |
|---|---|
|  **Agent** handles **Processing.** |  **Processing** requires **Instrument.** |