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.

consumption
New
consists
Change
Control
name
requires
participation
Modeling
Agent
symbol
Figure
Model
OPL
Physical
object
changes
Language
Semantics
left
Since
inzoomed
order
first
output
OPD
values
SysML
1
event
effect
Attribute
Part
sentence
Time
using
Parts
function
relations
Procedural
2
states
Crash
thing
Condition
input
value
use
Set
statespecified
c
B
relation
Vehicle
used
process
processes
Diagram
example
link
least
level
links
System
right
instance
must
ONE
Things
case
objects
OPM
P3
table
state
structural
also
Instrument
three
two
result
May
fork
Constraint
Systems
phrase
Class
subprocesses

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/