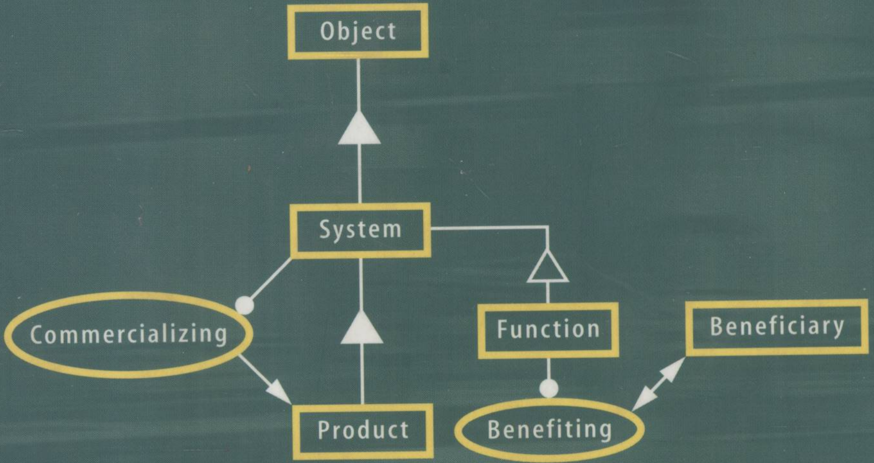
Dov Dori

Object-Process Methodology

A Holistic Systems Paradigm



with CD-ROM

Springer

Dov Dori

Object-Process Methodology

A Holistic Systems Paradigm

Foreword by Edward F. Crawley

With 246 Figures and CD-ROM

Springer

Professor Dov Dori

Technion, Israel Institute of Technology
Faculty of Industrial Engineering and Management
Haifa 32000, Israel

dori@ie.technion.ac.il

and

Massachusetts Institute of Technology
Engineering Systems Division, Building E40-347
77 Massachusetts Avenue
Cambridge, MA 02139-4307, USA

dori@mit.edu

Library of Congress Cataloging-in-Publication Data applied for
Die Deutsche Bibliothek – CIP-Einheitsaufnahme
Dori, Dov: Object process methodology: a holistic systems paradigm/
Dov Dori. – Berlin; Heidelberg; New York; Barcelona; Hong Kong;
London; Milan, Paris; Tokyo: Springer, 2002
ISBN 3-540-65471-2

ISBN 3-540-65471-2 Springer-Verlag Berlin Heidelberg New York

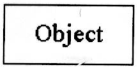
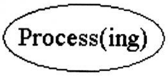
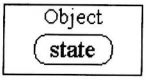
ACM Computing Classification (1998):
D.2.2, D.2.9–10, D.3.2–4, H.1.1, I.2.4

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH
http://www.springer.de

Material in this book is patent pending.
A patent titled "Modeling System" was filed with the US Patent & Trademark Office, Document Number **20020038206**.

Cover design: KünkelLopka, Heidelberg
Data conversion: G&U e.Publishing Services, Flensburg
Printed on acid-free paper   SPIN: 10560832   45/3142 GF– 5 4 3 2 1 0

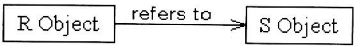
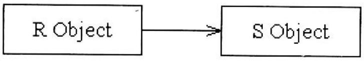
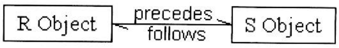
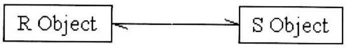
The Building Blocks

	Visual Representation	Textual Form	Definition	Description
Entities		Nouns; capitalized first letter in every word; if ending with "ing", "Object" is placed as a suffix	*An **object** is a thing that has the potential of stable, unconditional physical or mental existence.*	Static things. Can be changed only by processes.
		Nouns in gerund form; capitalized first letter in every word; if not ending with "ing", "Process" is placed as a suffix	*A **process** is a pattern of transformation that an object undergoes.*	Dynamic things. Are recognizable by the changes they cause to objects.
		Nouns, adjectives or adverbs; non-capitalized	*A **state** is a situation an object can be at.*	States describe objects. They are attributes of objects. Processes can change an object's state.
OPL Conventions	**Non-Reserved Words**	Arial bold by default	*Names of entities.*	Words used by the system architect, unique to the system.
	reserved words	Arial by default; non-capitalized	*Object-Process Language (OPL) words.*	Words or phrases used by OPL, the same in every sentence of a certain type.

Links: The Mortar

Tagged Structural Links

Generally used between objects, but may also be used between processes.
Cannot be used to link an object to a process.

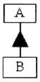
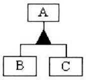
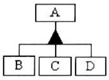
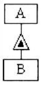
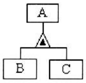
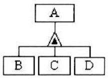
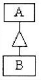
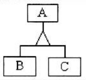
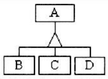
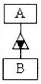
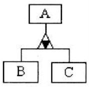
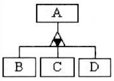
Link Name	Object Process Diagram (OPD) Symbol	OPL Sentence	Description
Tagged		**R Object refers to S Object.**	Relation from source object to destination object; relation name is entered by architect, and is recorded along link.
(Null)		**R Object** relates to **S Object.**	Relation from source object to destination object with no tag.
Bi-directional Tagged		**R Object precedes S Object. S Object follows R Object.**	Relation between two objects; relation names are entered by architect, and are recorded along link.
(Null) Bi-directional		**R Object** and **S Object** are related.	Relation between two objects with no tag.

Links: The Mortar (continued)

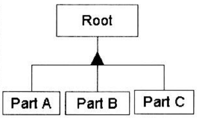
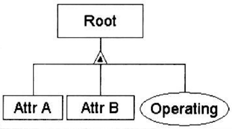
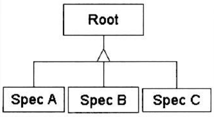
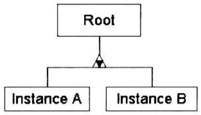
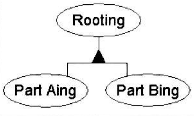
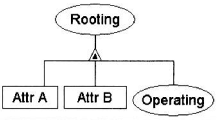
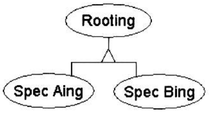
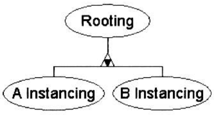
The Four Fundamental Structural Relations

Shorthand Name	Aggregation	Exhibition	Generalization	Instantiation
Symbol	▲	◬	△	⧩
Meaning	Relates a whole to its parts	Relates an exhibitor to its attributes	Relates a general thing to its specializations	Relates a class of things to its instances

A fundamental structural relation can have many descendants.
The different OPL sentences and OPD pictures are listed below.

Structural Relation Name and **Shorthand Name**	Number of Descendants						Description
	One		Two		Three or more		
	OPD	OPL	OPD	OPL	OPD	OPL	
Aggregation-Participation		**A** consists of **B.**		**A** consists of **B** and **C.**		**A** consists of **B, C,** and **D.**	B, C and D are parts of the whole A.
Exhibition-Characterization		**A** exhibits **B.**		**A** exhibits **B** and **C.**		**A** exhibits **B, C,** and **D.**	B, C and D are attributes of A. If B is a process, it is an operation of A.
Generalization-Specialization		**B** is an **A.**		**B** and **C** are **A**s.		**B, C,** and **D** are **A**s.	B, C and D are types of A.
Classification-**Instantiation**		**B** is an instance of **A.**		**B** and **C** are instances of **A.**		**B, C,** and **D** are instances of **A.**	B, C and D are unique objects of the class A.

The four fundamental relations are also applicable to processes. Only exhibition can link objects with processes. Instantiation cannot generate a hierarchy while the other three can. Any number of things can be linked to the root.

	Aggregation	Exhibition	Generalization	Instantiation
Object				
Process				

Object-Process Methodology

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

To my family

Foreword

The contemporary technological world is made up of complex electro-mechanical-information products that intimately involve human operators. Such a description includes products that range from a simple kitchen appliance to an air-traffic control system. Increasingly, the value these products deliver is based on their system nature; they have many interacting components that combine to produce emergent features or services that are desirable to the customer.

The development of more rigorous approaches to the engineering of these systems is seen by industry as a key area for process improvement, and should be recognized by academia as an important area of scholarly development. Since the engineering process dominates development time and cost, and the system engineering process defines value delivered by a product, improvements in system engineering and product modeling will have direct influence on product utility and enterprise competitiveness. The task of engineering a new system has become more complicated by the vastly 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 rigorously 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.

Object Process Methodology provides such a framework. OPM includes a clear and concise set of symbols that form a language enabling the expression of the system's building blocks and how they relate to each other. It is a symbolic representation of the objects in a system and the processes they enable. Considering 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 system engineering and product/system development.

We are now entering a phase, in which symbolic representation and manipulation can be developed. Such a symbolic system can lay the foundation for the ultimate step in the evolution of a discipline, the ability to predict the outcome beforehand. 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.

OPM represents the two things that are inherent in a system: its objects and its processes. This duality is recognized throughout the community that 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 recognizes the value of thinking about processes in parallel with objects. Not only does OPM represent both objects and processes, but it also explicitly shows the connections between them.

Object Process Methodology has another fundamental advantage – it represents the system simultaneously in a graphic representation and in a natural language. The two are completely interchangeable, and represent the 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 hard-wired into the human brain.

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

I have used OPM in my System Architecture course at MIT. 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. Incorporating OPM into 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.

One can anticipate that there will be many academic applications of OPM. I would consider using it in intermediate or advanced subjects in system engineering, product development, engineering design and software engineering. It is ideal for courses that demonstrate how various disciplines come together to form a multi-disciplinary product.

Likewise, OPM can form the backbone of a corporate or enterprise modeling system for technical products. 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 and only a few other modeling frameworks are available for decision support.

Massachusetts Institute of Technology
Cambridge, Massachusetts
May 2001

Edward F. Crawley

Preface

Before Fortran, putting a human problem ... on a computer was an arduous and arcane task. ... Like high priests in a primitive society, only a small group of people knew how to speak to the machine. Yet there were some heretics in the priesthood, and Mr. Backus was one of them. "I figured there had to be a better way," he recalled nearly five decades later, "you simply had to make it easier for people to program."

S. Lohr (2001)

This book is about how to make it easier for people to understand and develop systems. Systems are all around us. Understanding and developing complex systems involve a host of disciplines that need to be brought together in a unifying framework. Systems evolve over time, and they keep growing and getting more complex. Hundreds, perhaps thousands, of individuals are required to maintain and operate systems such as electrical power grids or globally interconnected telecommunication networks. Nature's global climate, the solar system and biological systems are at least as intricate. Systems science and engineering are emerging as new interdisciplinary fields of study that identify and utilize important commonalties among systems of all types.

A fundamental requirement of any science and engineering domain is that its intellectual underpinnings be formulated and well grounded. Such formulation, in turn, mandates that a set of elementary building blocks is agreed upon and relations among these building blocks are studied and understood. A unifying approach is indispensable for developing, communicating, supporting, and evolving systems of various domains, types, magnitudes and complexities. To this end, a clear and concise language must be developed. Pioneers in established science and engineering disciplines, like physics, chemistry, mechanical engineering, and electrical engineering, have long agreed on such languages. More recently, biology has joined this list and is now establishing the molecular foundations of life. Being a young filed, systems science and engineering is just beginning to contemplate this problem and sort it out. This book will hopefully advance the state of affairs regarding our ability to model and understand systems.

The book is intended for people interested in modeling systems, and reading it does not require special background in either mathematics or programming. Specifically, system integrators, analysts, designers, modelers, executives, and project leaders in a variety of industry and service domains, private as well as public, will benefit from reading the book and applying Object-Process Methodology (OPM) for the purpose of developing better systems faster and more reliably. An equally important point is that developing systems with OPM is fun. The combination of graphics and natural language that automatically complement each other when an automated tool, such as Systemantica® (Sight Code, 2001) is employed, is an enjoyable way of analysis and design that increases one's confidence in the quality of the resulting system specification. Information technology professionals, including computer scientists, software engineers, information systemmanagers, and database administrators will gain insight into possible extensions of Object technology. Scientists and engineers, along with science and engineering educators and students of all ages will hopefully find OPM useful to model, communicate, and explain systems they research and design.

The book, which consists of 15 chapters arranged in three parts, is designed as a textbook for graduate or advanced undergraduate courses. Typical course titles include engineering systems, specification and analysis of information systems, systems architecture, information systems engineering, systems design and management, or Object-Process Methodology. Indeed, various versions of this book have been used and tested in graduate and undergraduate courses at the Technion and MIT during the past five years. Each chapter concludes with a summary of the chapter's highlights and problems that enable hands-on experience, elaborate on concepts discussed in the chapter and provide food for further thought.

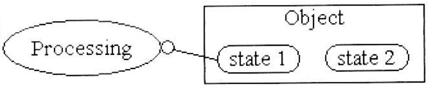
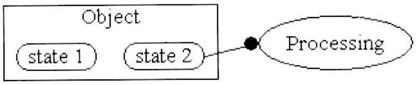
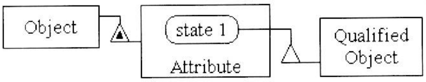
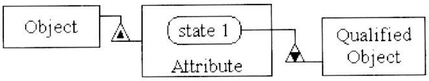
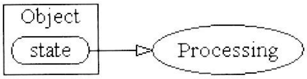
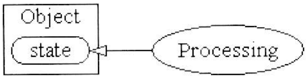
Part I, ***Foundations of Object-Process Methodology***, is a gradual exposure to OPM. Chapter 1 is a gentle introduction to basic principles of OPM through a simple example of a wedding. Walking through a comprehensive case study of an automated teller machine (ATM), Chapter 2 exposes the reader to Object-Process Diagrams as the graphic representation of the single object-process model. Continuing with the ATM case study, Chapter 3 introduces Object-Process Language as the complementary modality to the graphic one and shows how the two synergistically reinforce each other. Chapter 4 discusses in depth the two basic building blocks of OPM, objects and processes. In Part II, ***Concepts of OPM Systems Modeling***, the dynamic and static system aspects are the focus of Chapters 5 and 6, respectively. The next two chapters elaborate on the four OPM fundamental structural relations. Aggregation and Characterization are discussed in Chapter 7; Generalization and Classification in Chapter 8. Chapter 9, which concludes Part II, is devoted to complexity management. Part III, ***Building Systems with OPM***, shows how OPM is used to develop systems, products and projects. Systems and modeling is the topic of Chapter 10. Chapter 11 provides a comprehensive OPM model of system evolution and lifecycle, and discusses how this model can be used to develop systems in an OPM-based environment. The next two chapters expand upon issues presented in Parts I and II. Chapter 12 elaborates on states, while Chapter 13 presents advanced OPM concepts. Chapter 14 discusses systems theory and Chapter 15 concludes with a survey of object-oriented (OO) methods and their relation to OPM.

I thank Professor Ed Crawley for his encouragement and support throughout my stay at MIT and, in particular, for his enthusiastic Foreword to this book. OPM has become and integral part of the course "Systems Architecture," which he and Professor Olivier de Weck teach at MIT. Thanks to Robert M. Haralick whose draft, written at my request, was the basis for Section 4.1. Thanks for the professional editing as well as the detailed and helpful comments to Dr. Hans Wössner of Springer-Verlag. Thanks to Idan Ginsburg and William Litant for their meticulous proofreading. Thanks to my wife Judy and my daughters, Limor, Tlalit, Shiri and Inbal, who have been really helpful and considerate. Special thanks to Shiri Dori, whose thorough proofreading and wise comments, at both the editorial and content level, as well as the problems she composed, were extremely helpful.

Haifa, Israel, and Cambridge, Massachusetts Dov Dori
January 2002

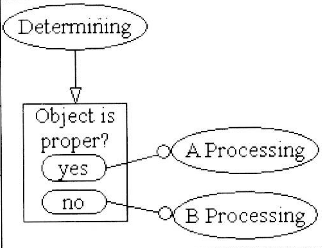
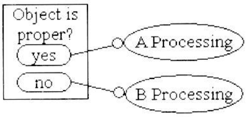
States (continued)

State-related Links

Link Name	OPD Symbol	OPL Sentence	Description
Condition		**Processing** occurs if **Object** is **state 1.**	Object is an instrument. It must be at a specific state in order for the process to occur.
Agent Condition		**Object** must be at **state 2** for **Processing** to occur.	Object is an agent. It must be at a specific state in order for the process to occur.
Qualification		**Qualified Object** is an **Object,** the **Attribute** of which is **state 1.**	Qualified Object is a type of Object. It must be at a particular state of Object's Attribute.
Instance Qualification		**Qualified Object** is an instance of an **Object,** the **Attribute** of which is **state 1.**	Qualified Object is an instance of class Object. It must be at a particular state of Object's Attribute.
State Specificied Consumption		**Processing** consumes **state Object.**	Process consumes object only if it is at a certain state.
State Specificied Result		**Processing** yields **state Object.**	Process creates object at a certain state.

Boolean Objects

Specialized informatical objects. Boolean objects are questions, and they always have two states (the answers): yes and no.

Link type	OPD Symbols	OPL Sentence	Description
Determination (a Result Link)	 Determining Object is proper? yes no A Processing B Processing	**Determining** determines whether **Object** is **proper.**	Process yields a Boolean object that poses a "yes or no" question. The process then determines the answer.
Condition link		**A Processing** occurs if **Object** is **proper.**	If the answer is "yes," a certain process occurs. If the answer is "no", a different process occurs.
Negative condition link		**B Processing** occurs if **Object** is not **proper.**	
Both condition links	 Object is proper? yes no A Processing B Processing	**A Processing** occurs if **Object** is **proper,** otherwise **B Processing** occurs.	Compound sentence: if the answer is "yes," a certain process occurs, otherwise a different process occurs.

Contents Overview

Table of Contents