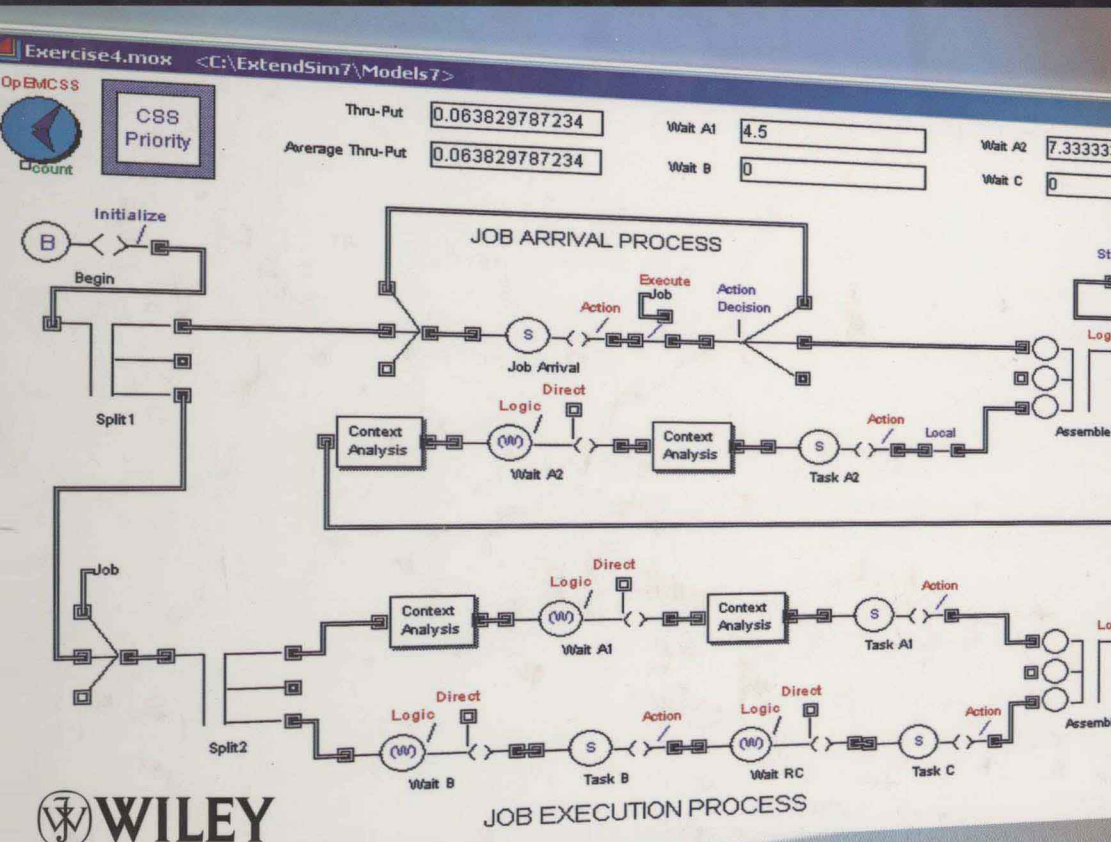
Wiley Series in Systems Engineering and Management • Andrew P. Sage, Series Editor

Simulation-Based Engineering of Complex Systems

Second Edition

John R. Clymer

CD-ROM INCLUDED



WILEY

SIMULATION-BASED ENGINEERING OF COMPLEX SYSTEMS

Second Edition

JOHN R. CLYMER
Professor of Electrical Engineering College of Engineering and Computer Science, California State University, Fullerton

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data is available.

Clymer, John R.
Simulation-based engineering of complex systems / John R. Clymer. – 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-40129-3 (cloth/CD)
1. System analysis–Simulation methods. I. Title.
QA402.C5347 2009
620.001'171–dc22

2008036199

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

WILEY SERIES IN SYSTEMS ENGINEERING AND MANAGEMENT

Andrew P. Sage, Editor

ANDREW P. SAGE and JAMES D. PALMER
Software Systems Engineering

WILLIAM B. ROUSE
Design for Success: A Human-Centered Approach to Designing Successful Products and Systems

LEONARD ADELMAN
Evaluating Decision Support and Expert System Technology

ANDREW P. SAGE
Decision Support Systems Engineering

YEFIM FASSER and DONALD BRETTNER
Process Improvement in the Electronics Industry, Second Edition

WILLIAM B. ROUSE
Strategies for Innovation

ANDREW P. SAGE
Systems Engineering

HORST TEMPELMEIER and HEINRICH KUHN
Flexible Manufacturing Systems: Decision Support for Design and Operation

WILLIAM B. ROUSE
Catalysts for Change: Concepts and Principles for Enabling Innovation

LIPING FANG, KEITH W. HIPEL, and D. MARC KILGOUR
Interactive Decision Making: The Graph Model for Conflict Resolution

DAVID A. SCHUM
Evidential Foundations of Probabilistic Reasoning

JENS RASMUSSEN, ANNELISE MARK PEJTERSEN, and LEONARD P. GOODSTEIN
Cognitive Systems Engineering

ANDREW P. SAGE
Systems Management for Information Technology and Software Engineering

ALPHONSE CHAPANIS
Human Factors in Systems Engineering

YACOV Y. HAIMES
Risk Modeling, Assessment, and Management, Third Edition

DENNIS M. BUEDE
The Engineering Design of Systems: Models and Methods, Second Edition

ANDREW P. SAGE and JAMES E. ARMSTRONG, Jr.
Introduction to Systems Engineering

WILLIAM B. ROUSE
Essential Challenges of Strategic Management

YEFIM FASSER and DONALD BRETTNER
Management for Quality in High-Technology Enterprises

THOMAS B. SHERIDAN
Humans and Automation: System Design and Research Issues

ALEXANDER KOSSIAKOFF and WILLIAM N. SWEET
Systems Engineering Principles and Practice

HAROLD R. BOOHER
Handbook of Human Systems Integration

JEFFREY T. POLLOCK AND RALPH HODGSON
Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration

ALAN L. PORTER AND SCOTT W. CUNNINGHAM
Tech Mining: Exploiting New Technologies for Competitive Advantage

REX BROWN
Rational Choice and Judgment: Decision Analysis for the Decider

WILLIAM B. ROUSE AND KENNETH R. BOFF (editors)
Organizational Simulation

HOWARD EISNER
Managing Complex Systems: Thinking Outside the Box

STEVE BELL
Lean Enterprise Systems: Using IT for Continuous Improvement

J. JERRY KAUFMAN AND ROY WOODHEAD
Stimulating Innovation in Products and Services: With Function Analysis and Mapping

WILLIAM B. ROUSE
Enterprise Tranformation: Understanding and Enabling Fundamental Change

JOHN E. GIBSON, WILLIAM T. SCHERER, AND WILLAM F. GIBSON
How to Do Systems Analysis

WILLIAM F. CHRISTOPHER
Holistic Management: Managing What Matters for Company Success

WILLIAM B. ROUSE
People and Organizations: Explorations of Human-Centered Design

GREGORY S. PARNELL, PATRICK J. DRISCOLL, AND DALE L. HENDERSON
Decision Making in Systems Engineering and Management

MO JAMSHIDI
System of Systems Engineering: Innovations for the Twenty-First Century

ANDREW P. SAGE AND WILLIAM B. ROUSE
Handbook of Systems Engineering and Management, Second Edition

JOHN R. CLYMER
Simulation-Based Engineering of Complex Systems, Second Edition

KRAG BROTBY
Information Security Governance: A Practical Development and Implementation Approach

CUSTOMER NOTE: IF THIS BOOK IS ACCOMPANIED BY SOFTWARE, PLEASE READ THE FOLLOWING BEFORE OPENING THE PACKAGE.

This software contains files to help you utilize the models described in the accompanying book. By opening the package, you are agreeing to be bound by the following agreement:

SIMULATION-BASED ENGINEERING OF COMPLEX SYSTEMS

Preface

The story of developing the maritime chronometer, which took place during the early eighteenth century, is a fascinating tale of creating a self-regulating system that was capable of keeping time accurate enough to navigate across broad stretches of ocean. Mariners had used the position of the sun and the stars for approximate navigation upon the sea for thousands of years; however, accurate determination of ship longitude, and thus ship position, depended on knowing the exact time of day to take measurements. This problem was solved by the invention of the self-regulating maritime chronometer, thus enabling the complete conquest of the world's oceans for trade and exploration. Clerk Maxwell [Maxwell, J. C., *Proc. Roy. Soc. (London)*, March 5, 1868] described the concept of a self-regulating governor. James Watts quickly implemented this concept as the now famous fly-ball form of speed control for steam engines, thus enabling the industrial revolution. This device served to keep the speed of the engine at a selectable level regardless of load by controlling the volume of steam available to the piston during each revolution. If the engine speed increased, the fly-balls fastened to a spinning shaft moved outward due to centrifugal force, thus moving an attached slider collar. The steam inlet valve, which was attached to the slider collar, was moved toward its closed position, thus preventing the engine from speeding up. Conversely, if the engine speed decreased, the fly-balls would relax, producing the opposite affect. The resulting dynamic stability produced by this ingenious feedback control devise was not due only to the mass of the fly-balls, or to the strength of the spring attached to the collar, or to the length of the lever arm that connected the collar to the steam valve, or to the efficiency of the engine, or to the load on the engine. It was due to the concurrent interaction of all of these components working together.

In 1948, Norbert Weiner (Weiner, N., *Cybernetics*, Wiley, 1948) popularized the term *cybernetics* by explaining how the foregoing control technology could be interrelated with computer and communication technologies to implement not only governors but also pursuit guidance systems and the early forms of goal-seeking systems.

In parallel, the field of general systems theory, which grew out of the field of general semantics, formalized the notion that a system consists of two or more elements having specific interrelationships and even relationships among the relationships. Further, that the overall response of a system to a given stimulus was rarely a simple linear function and was as much influenced by the nature of the relationships as by the properties of the elements. Out of these early beginnings came a flood of ideas and innovations that mold our modern world today and our thinking about complex systems.

Jay Forrester introduced systems dynamics models that clarified the ramifications of the structure of various feedback and feedforward relationships and of the affect of time delays on these relationships. Ross Ashby articulated the law of requisite variety, which says that the controller of a system has to have at least one degree of freedom greater than the system being controlled. Lotfi Zadeh's fuzzy logic generalized classical logic to allow classification of system features to be ambiguous. For example, confidences in feature facts specifying whether two objects are either close or far can both have a value that varies, depending on the distance between the objects. The classifier system block, discussed in Chapter 7, facilitates rule induction by minimizing ambiguity in each decision context during the generation of new rules.

Others noted that many relationships were not static but needed to be modeled as active logic ("SIMULA 67 Common Base Language," O. J. Dahl, B. Myhrhaug, K. Nygaard, Norwegian Computing Center, Forshning Veien 1B, OSLO 3 Norway, 1968) and, bilaterally, that human reasoning could be modeled such that computer technology could be leveraged to help do pattern recognition, choice making, generative planning, and even the design of machines that could design machines. These early software engineering and artificial intelligence efforts have evolved into the fields of multiagent systems and distributed, artificial intelligence (AI). Chapter 7 discusses how such human reasoning can be simulated with OpEMCSS.

Over these decades engineering educators have struggled to cope with such complex systems by "simplifying" them into piecewise linear continuous-time models and discrete-event queuing system models. But systems are not that easily decomposed. Although progress was made in modeling and simulating such systems, the skill levels required and the project times involved often meant that the design decisions were already made by the time the simulation was ready to provide much needed system analysis and evaluation input to the design process.

A new wave of understanding, and computer-based tools for even better understanding of complex self-synchronizing systems, was founded when a group of nuclear physicists interested in quantum theory came into relationship with a

group of economists interested in understanding the higher-order, implicit characteristics of stock markets and the global economy. Their dialog was greatly accelerated by John Holland, discussed in Chapter 7, and his suite of software tools and constructs for modeling implicit systems and simulating them to reveal their emergent behaviors, which were otherwise unpredictable by mathematicians and logicians. The field of complex adaptive systems (CAS) took form and has spawned a spectrum of studies.

I am indebted to all the complex systems researchers and innovators that have gone before and have influenced the development of the ideas, tools, and methods expressed in this book

Acknowledgments

I wish to thank everyone who provided me with assistance in writing this book: Bill Brown, Dave Brown, Dennis Buede, Philip Corey, Bill Cutler, Tony Genna, Mike Green, Sam Harbaugh, David Harris, Steve Helton, Kamran Iqbal, Carol Jacoby, Eckhart Linneweh, Peter Macgregor, Jim Manson, Jack Ring, Jim Van-Gaasbeek, and Dave Watt. Special thanks go to my students Jose Garcia and Linda Gregory and to my colleague and friend Jack Ring for suggesting the historical overview presented at the beginning of the preface. I also want to acknowledge the contribution of employment with FORELL Enterprises (Buena park, CA), Rockwell International (Anahiem, CA), Navy Fleet Analysis Center (Norco, CA), and General Electric Company (Phoenix, AZ) to the development of the methodology presented in this book.

Overview

I have been a researcher, teacher, and practitioner of systems engineering for 40 + years, including system design, systems analysis and evaluation, and systems science of complex systems. During this time, the computer, transportation, manufacturing, business, and military systems that I have worked on were discovered to be best characterized as a set of interacting, concurrent processes. This discovery inspired the development of context-sensitive systems (CSS) theory as a way of thinking about interacting concurrent processes. Also developed was the graphical modeling language, Operational Evaluation Modeling (OpEM), to express CSS models of both existing and conceptual systems. Further, a graphical discrete-event simulation library, OpEMCSS, was developed more recently to enable rapid development of CSS models and simulations in the OpEM language. I believe that the CSS theory, OpEM modeling language, and OpEMCSS library, described in this book, can be applied to understand complex adaptive systems (CAS) and to perform simulation-based systems engineering (SBSE).

Simulation-based systems engineering mitigates system development problems (resulting from "stove-piped systems" design methods) that are caused by the failure to optimize the interoperability and synergisms among all component algorithms and methods at the overall system level. Further, the interactions of the system with its external systems and the dynamic demands of the operational environment on the system must be included in a SBSE-level model.

An OpEMCSS-level model provides the structure and ontology (top-level formalisms) needed to connect detailed component models for SBSE. The SBSE approach is:

1. Apply the OpEM top-down systems design methodology discussed in this book.

2. Perform system concept and top-level design trade-offs to optimize stakeholder requirements using OpEMCSS.
3. Produce a systems design specification that includes component interface and qualification system requirements using a design capture database tool.
4. Develop component detailed models of alternative component algorithms and methods using the OpEMCSS special blocks (see Appendix C).
5. Perform virtual systems integration and system verification & validation (V&V) using the system-level OpEMCSS simulation.
6. Determine impact of requirement changes and conduct detailed design trades using the system-level OpEMCSS simulation.

There are many kinds of models used to develop simulations during SBSE. These models include functional flow, entity relationship, semantic networks, and various kinds of block diagrams to mention a few. However, only executable models can be used during SBSE because only executable simulation models, such as OpEMCSS simulations and Markov models, can be used to numerically evaluate design trade-offs and derived requirements.

The OpEMCSS library works with the popular commercial software tool, ExtendSim (a registered trademark of Imagine That Inc. of San Jose, California), which was chosen for two major reasons. First, the ExtendSim LT-RunTime student version is provided on a CD in the back of the book to be used in the many hands-on experiments found in the book (see Appendix A). Second, it is a powerful simulation tool when expanded by OpEMCSS. I have programmed both continuous-time, discrete-event, and hybrid computer simulations in FORTRAN, Pascal, and C/C++, and I have studied various ways to make such programs execute faster and more efficiently. In my opinion, ExtendSim is a very efficient implementation of a high level, graphical icon based, simulation tool. The ExtendSim + OpEMCSS icon blocks automatically provide more than 95% of all simulation code that in the past had to be programmed by hand. In context-sensitive systems, these programming details are very complex and would otherwise require extensive programming skill and effort to accomplish. ExtendSim, with the OpEMCSS library, gives students and systems practitioners the ability to experiment with complex, context-sensitive interactions and quickly build a model. Time is not wasted dealing with complex programming details and writing extensive code, but rather the emphasis is on complex systems design, analysis, and evaluation for SBSE.

I believe the best way to learn the basic principles of complex systems, discussed throughout this book, is to actually experience them directly through hands-on experimentation. Throughout the text, the reader is invited to do simulation-based experiments that demonstrate the principles of complex systems. The reader is constantly engaged through the examples in the book to actively learn and participate. I believe that a student that learns this way is more likely to be able to apply what he or she has learned in diverse fields and extend that knowledge to new fields and new problems.

Understanding complex systems is like peeling an onion; one does it layer by layer. This book is organized to begin with a simple system that the reader already understands and build on this understanding through a series of examples. Each example has a lesson about complex system or SBSE principles to teach. Gradually, the book introduces more and more complex system principles. It is important that students and other readers study each of the models in the book and do the exercises in order to gain an understanding of the basic principles that each model represents. Thus, the book is designed for a broad spectrum of people to gain an understanding of complex systems and SBSE. It will be shown that, although complex systems have behaviors that are difficult to understand, the OpEMCSS modeling building blocks are simple to use and easy to understand.

COMPLEX ADAPTIVE SYSTEMS

All complex adaptive systems have emergent behaviors that result due to the interactions of their components. Such system-level behaviors only occur if components are working together; they do not occur when operating any single component alone. Thus, we cannot understand each component as it operates independently to gain an understanding of the whole system.

Often the emergent behavior of the system is not predicted when a system concept is proposed, and its occurrence is a surprise when the system concept is built. This is why simulation of the entire system, operating within its operational environment, is an important part of engineering of complex systems discussed in this book.

In CAS, I believe the main interaction among concurrent processes is communication and adaptation. There are three main kinds of interactions discussed in this book, but the communication and adaptation interaction is the one that results in changes in the system or environment that propagate into the future through various causal paths. These causal paths through time dictate the behavior of individual system components, and the result is emergent behavior as discussed above. Because of these communication and adaptation interactions, the whole system is more complex than can be predicted through analysis of each system component separately.

As an example of a system having emergent behavior, a distributed vehicle traffic control network located in a large city is discussed. This traffic control network is an example of a system of systems (SOS) where each system in the network independently provides specific services, and each system can operate independently of the rest of the SOS. Additional services are provided through collaboration among the networked systems. Network centric operation (NCO) of related business units and combat system platforms are other examples of SOS that are currently of research interest.

Each major intersection has a vehicle traffic light controller to determine traffic light timing. In this system, each traffic light controller uses its perceptions about incoming traffic flow to optimize light timing, thus minimizing local vehicle

waiting time. The result of each traffic light controller adapting light timing to accommodate traffic flow coming from other intersections is to minimize the average waiting time in the entire network. Global minimization of traffic waiting time results as a consequence of the emergent behavior of this system, which is the self-synchronization of each traffic controller's light timing with other controllers.

As light timing control in the overall traffic grid evolves in the way discussed above, a complex but definite pattern in network operation, north–south red-to-green transition times, emerges out of an initial random light pattern. The emergent behavior of the traffic grid cannot be explained through an understanding of each controller alone. Understanding only comes when we study the interactions of the controllers as they adapt their behaviors in response to perceived information about incoming traffic flow, achieving self-synchronization of all traffic light controllers in the network.

The desired emergent behavior for the traffic control system was achieved by experimenting with feedback in the network. Trying different sets of fuzzy control rules and control system gains did this. With no feedback, the traffic lights operate independently and average vehicle waiting time is high. With very strong feedback, the traffic control network operation appears chaotic (little self-synchronization occurs), and the average vehicle waiting time is high. When the feedback is just right, the emergent behavior discussed above is observed, and the average vehicle waiting time in the network is minimized. It is also interesting to note that the overall system never reaches steady-state operation; indeed, the system seems to be in a constant state of flux as observed by the random appearance of the light timing control signals, even when the average vehicle waiting time is being minimized, implying some kind of convergence.

INTENDED AUDIENCE AND PREREQUISITES

This book is intended as a senior elective and graduate-level textbook (a solution manual of the problems is available) and a practitioner's reference book. It is mainly intended for systems engineers, integrated product engineers, software engineers, industrial/manufacturing engineers, business management design people, and intelligent enterprises (IE) people.

However, I believe that this book can be used in any field that is concerned with collaborating/conflicting entities that perform a set of tasks that lead to satisfaction of a measurable goal that may or may not be explicitly known or stated. Such fields include, in addition to those mentioned above, societal systems and sociology, biological and ecological systems, economic systems, and others. There is currently no book known to me that covers the basic principles of complex systems in such a way that students and practitioners can readily extend these principles to all kinds of systems through hands-on experimentation using an icon-based simulation tool.

As will be demonstrated in Chapter 1, the prerequisite for this book is a basic problem-solving ability, which I believe the vast majority of people possess as

common sense. Each one of us formulates goals for ourselves all the time to solve problems confronting us. Once we state our goal, we visualize a set of tasks or steps that takes us from our current situation to one satisfying our goal. If we can expect help from others to execute these tasks, we organize the tasks into sequential and concurrent arrangement. We call our organized set of tasks, more commonly, a plan. The execution of our plan by a group of people hopefully leads to goal satisfaction. Such a plan can be developed using a task flow diagram and modeled as a collection of interacting concurrent processes, which are subjects of this book. A computer simulation program based on these interacting concurrent processes can be used to optimize our plan and make it robust in the presence of varying contingencies.

This book is not intended to be a survey of simulation in which simulation is an end unto itself. There are plenty of books already that do that. In this book, simulation is the means to an end; understand, design, analyze, and evaluate complex systems in order to do SBSE. However, I do discuss how simulations work, including basic principles such as what are the different kinds of simulation methods and what are some of the various applications of simulation during the engineering of complex systems.

This book is not intended to be a survey of complex adaptive systems either. There are many anecdotal-type books already published that provide insights into CAS. This book simply intends to provide hands-on learning and experimentation with the basic principles of complex systems and to teach simulation-based problem-solving skills and system design, analysis, and evaluation of SBSE methodology.

ORGANIZATION

Throughout this book a commonsense approach to understanding, designing, analyzing, and evaluating complex systems and doing simulation-based systems engineering is presented. It begins by discussing traditional ways of thinking about systems and then shows how one of these views, the operational view of systems, can be best expressed using interacting concurrent processes. A graphical language (OpEM) is presented that provides a natural way to describe interacting concurrent processes and implement them using simulation. A graphical discrete-event simulation library (OpEMCSS) is discussed throughout this book that implements this graphical language and provides a means to experiment with complex systems and do SBSE. A large number of example models are described that illustrate how to use the OpEMCSS library blocks to model complex systems. These examples are presented so that a large spectrum of readers can understand them.

In Chapter 1, general systems are defined and some of the more commonly used system models are described. Next, readers are asked to analyze a goal-oriented activity they already know how to do and to build a functional flow model of this activity. I introduce some of the basic OpEMCSS simulation