Computational Analysis, Synthesis, and Design of Dynamic Systems

Discrete-Event Modeling and Simulation

A Practitioner's Approach

Gabriel A. Wainer

CRC Press
Taylor & Francis Group

Discrete-Event Modeling and Simulation

A Practitioner's Approach

Computational Analysis, Synthesis, and Design of Dynamic Models Series

Series Editor

Pieter Mosterman

The Mathworks
Natick, Connecticut

Discrete-Event Modeling and Simulations: A Practitioner's Approach, *Gabriel A. Wainer*

Discrete-Event Modeling and Simulations: Theory and Applications, *Gabriel A. Wainer and Pieter Mosterman*

Model-Based Design for Embedded Systems, *Gabriela Nicolescu and Pieter Mosterman*

Multi-Agent Systems: Simulation & Applications, *edited by Adelinde M. Uhrmacher and Danny Weyns*

Discrete-Event Modeling and Simulation

A Practitioner's Approach

Gabriel A. Wainer

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-1-4200-5336-4 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Wainer, Gabriel A.
Discrete-event modeling and simulation : a practitioner's approach / Gabriel A. Wainer.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-4200-5336-4 (hardcover : alk. paper)
1. Computer simulation. 2. Discrete-time systems. I. Title. II. Series.

QA76.9.C65W35 2009
003'.3--dc22 2008039739

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

SECTION 2 Building Simulation Models: The CD++ Toolkit

SECTION 3 Applications

SECTION 4 Simulation and Visualization

Foreword

Modeling and simulation (M&S) is finally coming into its own as a discipline, a technology, and an industry. The need is now evident for a textbook that can serve as an introduction to the field that is in the process of "becoming." Such a text has to lay out the foundations of M&S in a clear and concise manner without having to provide the rigor that was needed in the theory as it was developed because this theory is accessible to readers in the original works. Further, the text should offer a wide variety of applications that suggest the unique power of M&S to tackle the kinds of complex, multidisciplinary problems that are increasingly demanding global attention.

The book that you have before you admirably satisfies the preceding criteria. It is divided into sections on introductory background, applications, and technical exposition. The background section first introduces the key concepts in M&S from the point of view of the classic text *Theory of Modeling and Simulation,* as well as other foundational texts. It then goes on to present the basic concepts of discrete event system specification (DEVS) and CD++, the author's implementation of a cellular space DEVS simulation environment. The application section is notable for its wide variety of examples covering physical and life sciences, business, and engineering domains—all developed within the CD++ framework. The technical exposition covers areas such as simulator construction and execution, visualization of simulation output, and the like.

As a teacher, you will be well supported by this book. I suggest that you start with the introductory chapters and then select one application area upon which to base a detailed exposition of the concepts in a form that students can more easily grasp. The area chosen should depend on the backgrounds that students bring with them from their academic degrees. After setting this foundation, you can discuss or have students read other examples from the panoply available in the book. I would be sure to have them start on projects to apply the methodology to areas of interest to them. Finally, I suggest going on to cover some of the more technical issues to a depth that corresponds to the students' interests and backgrounds.

Although the book is primarily intended for graduate students, it can also be used in upper-level undergraduate courses. The availability of the CD++ software, with its graphical and visualization support, is a major advantage enabling students to graduate with the necessary skills to make creative and productive contributions to the exciting emergence of the modeling and simulation discipline, technology, and industry.

Bernard P. Zeigler
Tucson, Arizona

Preface

Scientists, engineers, and practitioners of many professions have long relied on the creation of models to understand the phenomena they study. Traditional mathematical methods (i.e., differential equations) have been used for centuries as the main tool for analysis, comprehension, design, and prediction for complex systems in varied areas. However, these methods appeared unsuitable for studying the complex human-made systems developed during the twentieth century. In the same vein, the many models applied to the study of natural systems have brought about scientific knowledge that, in turn, has posed new, complicated questions that could not be answered by analytical methods.

The emergence of digital computers provided alternative methods of analysis for both natural and artificial systems. Since the early days of computing, users translated their analytical models into computer-based *simulations* (i.e., the execution of those models with particular sets of data using a computing device). This approach allowed solving problems with a level of complexity unknown in earlier stages of scientific development. Computer-simulated models also have additional benefits: they can be executed safely, and experiments can be easily repeated in a cost-effective, risk-free environment and are thus well suited for training purposes.

Computational methods based on differential equations could not be easily applied in studying human-made dynamic systems (e.g., traffic controllers, robotic arms, automated factories, production plants, computer networks, VLSI circuits). These systems are usually referred to as *discrete-event systems* because their states do not change continuously but, rather, because of the occurrence of events. This makes them asynchronous, inherently concurrent, and highly nonlinear, rendering their modeling and simulation different from that used in traditional approaches.

In order to improve the model definition for this class of systems, a number of techniques were introduced, including Petri Nets, Finite State Machines, min-max algebra, Timed Automata, etc. The classic bibliography of this field (which includes, for instance, references 1–3) and other advanced literature (for instance, references 4–6) have thoroughly discussed these and related techniques in detail.

In this book we will mainly focus on one of the existing theories of modeling and simulation called DEVS (discrete-event system specification) [7,8]. Defined by B. Zeigler in the 1970s, the DEVS formalism allows the modular description of discrete-event models that can be integrated using a hierarchical approach.

Although the classic literature on discrete-event systems thoroughly covers theory and methods, there is a need to bridge theory and practice, allowing practitioners in different domains to create discrete-event models and simulations easily. Although many domain experts (and newcomers to the field of modeling and simulation) might know the basics about modeling and simulation theory (including discrete-event methodologies), they usually lack the skills and expertise to create the advanced models needed to study interesting problems. Thus, the focus of this book is to bridge this gap between theory and practice for discrete-event systems.

In order to achieve our goals, we rely on the use of the CD++ modeling and simulation environment [9], an open-source framework that enables simulation of discrete-event models (with specialized support for cellular models). Although we introduce the generics of the underlying theory, we focus on the creation of a variety of models in different areas of interest while giving details on how to create advanced simulation engines to execute these models. We also show how to build independent graphical user interfaces that have been built as open-source (and using varied three-dimensional rendering technologies). Experts in modeling and simulation can use this book as a

companion to the theoretical literature in the field, and the practitioners can focus on the practical aspects and the example applications.

The book is organized in four different parts. The "Concepts" part gives a general perspective on discrete-event modeling and simulation and a brief description of the DEVS and Cell-DEVS formalisms. Part II, "Building Simulation Models: the CD++ Toolkit," introduces the definition of DEVS models using CD++. This part starts by introducing the features of CD++, showing how to create basic models with the tool. It then concentrates on models of generic discrete-event systems, followed by a description of how to map different modeling techniques to DEVS (i.e., Petri Nets, Finite State Machines, Bond Graphs, Modelica, Layered Queuing Networks, and VHDL). We show several examples of the definition of such techniques in DEVS and discuss how to implement them in a discrete-event simulator such as CD++. Part III, "Applications," starts with a chapter on biology and medicine and then moves to defense and emergency planning, after which it discusses architecture and construction, environmental sciences, physics and chemistry, artificial systems, and urban traffic. The last part of the book, "Simulation and Visualization," elaborates on the creation of simulation software for DEVS models and analyzes the creation of three-dimensional visualization environments associated with these tools.

A variety of resources has been made available online to be used by the reader. Basic information on CD++ can be found at http://cell-devs.sce.carleton.ca, where the reader will find a complete user manual, installation tools, and the software application. It also includes links to an open-source version of the software (interested developers are encouraged to participate in the development of this project). All the examples discussed in the book can be found in a repository of models available for general use, and we encourage the reader to use such models for learning. Examples in the book are simplified versions of the actual versions found online. The examples focus on the main ideas and skip some of the details in order to make them more understandable by the reader; each model in the repository includes the complete software implementation and extended documentation for the reader.

We intend for the materials introduced to provide the reader with a wide variety of practical experiences, ranging from the creation of models, simulators, visualization tools, and theoretical analysis up to a large number of models for experimentation and open-source tools ready to use and modify. In order to show the feasibility of this approach, we include many examples developed by nonexperts. This allows readers to become familiar with the experience of others with a similar level of expertise, while providing the opportunity to improve and adapt other people's work to their own needs and interests.

This book is the result of years of collaboration with many students and colleagues. The list is so extensive that a separate section of acknowledgments has been included. Nevertheless, I would like in particular to thank Bernie Zeigler, Norbert Giambiasi, and Claudia Frydman (for their constant support) and Pieter Mosterman (for giving the initial stimulus). I owe everything to the love of my family. Carlos and Jacobo Wainer taught me the immense value of books from a very early age. Noemi and Maria Luisa taught me how to read them even earlier. Diana and Ian (who suffered the consequences of those teachings) gave me the most incredible lesson about love and patience every day I spent writing this book.

Gabriel A. Wainer
Ottawa, Canada

REFERENCES

1. Cassandras, C. G. 1993. *Discrete event systems: Modeling and performance analysis.* Homewood, IL: Aksen: Irwin.
2. Banks, J., J. S. Carson, B. L. Nelson, and D. Nicol. 2005. *Discrete-event system simulation,* 4th ed. Upper Saddle River, NJ: Prentice Hall.
3. Law, A. M., and W. D. Kelton. 2000. *Simulation modeling and analysis,* 3rd ed. Boston: McGraw–Hill.
4. Fishwick, P. A. 1995. *Simulation model design and execution: Building digital worlds.* Englewood Cliffs, NJ: Prentice Hall.
5. Cellier, F. E., and E. Kofman. 2006. *Continuous system simulation.* New York: Springer Science+ Business Media.
6. Toffoli, T., and N. Margolus. 1987. *Cellular automata machines: A new environment for modeling.* Cambridge, MA: MIT Press.
7. Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation,* 2nd ed. New York: Academic Press.
8. Zeigler, B. P. 1976. *Theory of modeling and simulation.* New York: Wiley-Interscience.
9. Wainer, G. 2002. CD++: A toolkit to develop DEVS models. *Software Practice and Experience* 32:1261.

The Author

Gabriel Wainer received the MSc (1993) and PhD degrees (1998, with highest honors) from the University of Buenos Aires (Argentina) and Université Paul Cézanne (Aix-Marseille III, France). In July 2000, he joined the Department of Systems and Computer Engineering, Carleton University (Ottawa, Ontario, Canada), where he is now an associate professor. He has held positions at the Computer Science Department of the University of Buenos Aires and visiting scholar positions in numerous places, including the University of Arizona, Ecole Polytechnique de Marseille, CNRS, University of Nice, and INRIA Sophia-Antipolis (France). He is the author of a book on real-time systems and another on discrete-event simulation and of more than 190 research articles. He has collaborated in the organizing of numerous conferences in the area.

Dr. Wainer has been principal investigator of different research projects (funded by the National Science and Engineering Research Council of Canada, Precarn, Usenix, the Canadian Foundation of Innovation, CANARIE, and private companies, including Hewlett-Packard, IBM, Intel, and MDA Corporation). He has been the recipient of various awards (including the Carleton University Research Achievement Award, IBM Eclipse Innovation Award, and the Leadership Award by the Society for Modeling and Simulation International—SCS, and CICC, Japan). Dr. Wainer is a member of the real-time and distributed systems lab at Carleton University and the head of the Advanced Real-time Simulation lab within Carleton University Center for Advanced Visualization and Simulation (V-Sim).

He is Special Issues editor of *Simulation Transactions of the SCS* and associate editor of the *International Journal of Simulation and Process Modeling.* He has been a member of the board of directors of the SCS and a chairman of the DEVS standardization study group (SISO). He is director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet.

Dr. Wainer's current research interests are related to modeling methodologies and tools, parallel/distributed simulation, and real-time systems.

Acknowledgments

The work presented in this book has been the result of the efforts of numerous students and collaborators. Although they have been acknowledged in the references and the authors of the models can be found on each model file, here we include the list of individuals who participated in the creation of the different tools and models.

Qi Liu, Amin Hammad, Hui Shang, Ezequiel Glinsky, Rami Madhoun, Dorin Petriu, Shaylesh Mehta, and Hesham Saadawi co-authored different sections.

Different collaborators have participated in the discussions that led to the creation of some of the work presented here, including some the applications: James Cheetham, C. Anthony Hunt, Michael Jemtrud, Ernesto Kofman, Tofy Mussivand, and Trevor W. Pearce. Other colleagues whose thoughts and discussions were of influence include Rod Bain, Olivier Dalle, Chris Herdman, Tag Gon Kim, Gabriela Nicolescu, James Nutaro, Tuncer Ören, Glen Ropella, Mamadou K. Traoré, Hans Vangheluwe, and Yuhong Yan.

Many students were fundamental in the creation of the software tools, in particular, Javier Ameghino, Amir Barylko, Jorge Beyoglonian, Shannon Borho, Wenhong Chen, Gastón Christen, Juan Cidre, Mariana D'Abreu, Alejandra Davidson, Alejandra Díaz, Alejandro Dobniewski, Ezequiel Glinsky, Marcelo Gutiérrez-Alcaraz, Christian Jacques, Kiril Kidisyuk, Qi Liu, Mariana Lo Tártaro, Alejandro López, Rami Madhoun, Alexandre Muzy, Jan Pittner, Daniel Rodriguez, Hui Shang, César Torres, Alejandro Troccoli, Verónica Vázquez, Wilson Venhola, and Yinfeng Yu. Students in our lab who collaborated in the applications include Khaldoon Al-Zoubi, Rodrigo Castro, Rachid Chreyh, Bo Feng, Rhys Goldstein, Shafagh Jafer, Leandro de San Miguel, Ayesha Khan, Mohammad Moallemi, Emil Poliakov, and Hesham Saadawi.

Finally, a large number of students created the models presented in the book and found in the model's repository. A noncomprehensive list includes T. Adams, A. Adidharma, M. Ahmed, B. Al-Aubiydy, D. Altman, B. Balya, A. Baranek, P. Barletta, J. Barrionuevo, A. Bender, P. Bendersky, X. Bing, Y. Boiko, M. Braunstein, D. Brignardello, M. Brunstein, A. Calvo, A. Campbell, J. Cao, S. Chao, J. Chazal, L. Checiu, A. Corvetto, P. Cremona, S. Daicz, F. Dellasoppa, L. De Simoni, C. Delannoy, P. Demidoff, A. Dias, W. Ding, C. Diuk, R. Djafarzadeh, M. El-Salam, A. Elshafei, S. Enrique, L. Fal, U. Farooq, E. Fernandez Rojo, J. Galaski, C. Gnanapragasam, A. Gonzalez, D. Grinberg, J. Hayes, M. Hussein, F. Iñón, R. Kirkner, R. Klett, K. Lam, S. Leon, L. Li, C. Lim, S. Lombardi, J. Long, M. MacLeod, P. MacSween, V. Mahendran, M. Mansour, S. Mehta, C. Miracola, A. Monadi, O. Muhi Kanwar, M. Núñez Cortes, J. Ogasawara, R. Ortiz, H. Pang, T. Pendergast, F. Petronio, D. Petriu, J. Pittner, M. Polimeni, L. Quinet, N. Rehman, M. Ricillo, D. Rubinstein, S. Sim, P. Sor, W. Sun, G. Thezier, S. Thurairasa, K. W. Tsui, G. Vasconcelos, D. Wassermann, P. Wu, X. Wu, P. Yeon, K. Yonis, R. Youssef, Y. Zhang, C. Zhang, X. Zhao, T. Zheng, and S. Zlotnik.

Section 1

Concepts

1 Modeling and Simulation Concepts

1.1 INTRODUCTION

Human beings are resourceful and curious. The need to explore, analyze our surroundings, and solve problems is in our nature (even from a very young age). There are many different reasons why we do these things, including improvement of our quality of life (e.g., by creating devices to reduce efforts or improve our safety), thirst for knowledge (e.g., in learning how plants grow), or even just for fun.

The first technique humans use to learn about (and maybe change) their environment is *experimentation* (for instance, if we want to learn how much clay and water must be mixed to do pottery, we conduct different trials until the desired consistency is obtained). Experimentation was the only way humans had to learn about their environment for thousands of years, and it is still one of the principal methods employed in problem solving (and a crucial part of child development). Figure 1.1 shows a basic scheme for experimentation.

There are two objects under consideration: the entity under study and the experimental frame (EF), which defines the conditions for experimentation. The EF defines not only how we experiment on the entity but also how we obtain the experimental results. In our pottery example, the entity is the mix of water and clay (that we can mold and then solidify in an oven), and the EF is the set of experiments done. Each experiment would be a different trial to mix clay and water (including different percentages of each material, varied temperatures of the mix, duration of the experiment, etc.). The experiments' results would include the consistency and texture expected and obtained for each different clay mix.

EXAMPLE 1.1

Let us suppose that we want to study the best possible allocation of desks in a classroom, in order to reduce energy costs. We need to decide where to put the desks and the heating/air conditioning sources. An experimentation-based solution would take different groups of students in different positions and would use a sensor (i.e., a thermometer) to measure the temperature in different areas in the classroom. In this example, the *entity* is a classroom and its temperature under different desk configurations. The *EF* is defined by the multiple student configurations (*experiments*) and the kinds of results expected at the end of each experiment (in this case, temperature of a room between –20 and +45°C). The *results* are provided by the thermometers used to measure the temperature. As we can see, the EF allows us to consider what the objectives of the experiment are (measure temperature in the room; the number of students or their weight is not of interest) and any assumptions we have about the experiment (do we use a digital thermometer or an analog one? One or many? Are we interested in fractions of a degree?).

Unfortunately, the problems we need to tackle are usually much more complex than learning how to mix the materials for making pottery or measuring the number of students and the temperatures in a classroom. In many cases, experimentation is not a feasible solution due to ethics, risks (e.g., we cannot study spread of an epidemic or fire evacuation in the classroom), or cost (we cannot study every possible configuration in the classroom because scheduling a study with a large number of

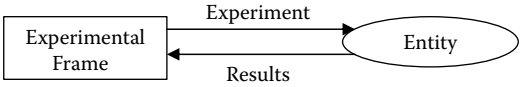

FIGURE 1.1 Problem solving through experimentation.

individuals for a long time can be costly). In other cases, experimentation is simply not possible (for instance, we cannot manipulate a star to understand its gravitational field; we cannot experiment on our classroom if the building still does not exist). However, in many cases, this kind of analysis is what we need.

Humans have found different ways of dealing with these issues. One of them is to abstract from the problem itself and then reason about it using a *model* of the problem. You might have started using this technique while thinking about Example 1.1, creating a mind picture of the room, students, sensing and heating devices, etc. You might even have started thinking about different student distributions and mechanisms to improve the heating according to their location, window positions, and building orientation. You might have sketched your ideas on paper, even using a scale model. Humans are well prepared (and trained from childhood) to create these models in a very natural way; they help us to think better about the problem we want to study.

Although different modeling techniques have been proposed, during the last 300 years, the *differential equation* formalism proposed by Newton and Leibniz has been the tool of choice for modeling and problem solving [1]. Differential equations provide a formal mathematical method (sometimes also called an analytical method) for studying the entity of interest. For instance, in Example 1.1, we could try to use Fourier's law [2] to study the temperature on the classroom's floor. These equations define the conduction of heat in a one-dimensional steady state isotropic as

$$q_x = -k\frac{\partial T}{\partial x}$$

Here,

q_x = the heat flux;
k = the thermal conductivity of the material under study (W/m°C);
T = the temperature field in the medium; and
$\partial T/\partial x$ = the temperature gradient (the minus sign indicates that the direction of heat flux is opposite the direction of increasing temperature).

When we try to solve problems through formal modeling (using, for instance, differential equations), the scheme previously presented in Figure 1.1 must be extended as shown in Figure 1.2. In this case, we can still do experimentation to obtain data about the entity under study, and we use such data to create a model of the entity (using differential equations or other analytical techniques). We use the model's experimental frame to put a context on the model's creation, indicating the kinds

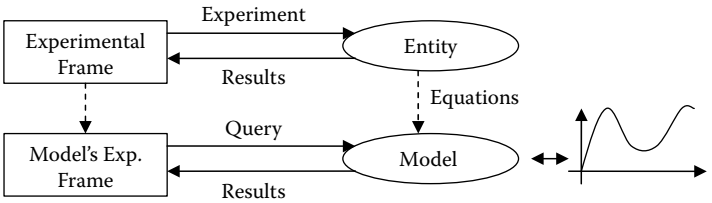

FIGURE 1.2 Problem solving through analytical modeling.

of questions we can ask and the assumptions we have made. For instance, in Example 1.1, we cannot find the average weight of the students (as we have decided to apply Fourier's equations), and we have assumed the use of a one-dimensional equation to approximate the temperature in the room. The model's EF also tries to mimic the experiments carried out on the original entity to obtain the desired results. For instance, in Example 1.1, we want to sense temperatures (we are not interested in social interactions between class students) and the effect of the number of students and their distribution in the room's temperatures. The results observed can be used, in turn, to modify the original entity (in this case, the classroom under study). Using this approach, we might even skip the initial experimentation step, just theorizing about the system's behavior through a pure model (even without any experimental data). In these cases, as soon as experimental data are available, they can be used to validate or reject the proposed theories.

These problem-solving techniques are analytical in the sense that they are symbolic and based on reasoning, and they try to provide general solutions to the problems to be solved. The idea is that we abstract what we learned about the entity into a model that represents the entity under study. This abstraction implies a loss of information, but it allows us to describe the behavior of the entities, analyze it, and prove properties of the proposed model (for instance, controllability and stability in the control system used to keep the room's temperature). The idea is that, if we are able to solve the equations, we will know the results of every possible experiment to be carried out on the entity. The solution is built using inference rules (which should be correct in the paradigm chosen to describe the model). If we need to obtain particular solutions, we just replace the symbolic values with their corresponding numerical counterparts. In Example 1.1, the Fourier equation allows us to find the solution to the problem for every value of x; if we are able to solve the equation, we will know the exact temperature in every single point (with infinitesimal scale).

Figure 1.2 also introduces an important concept: the behavior of the analytical model should match the one observed in the original entity. In this case, we say that the results given by the model are *valid*. To ensure this, a *validation* phase is required to check that the results given by the model match what we see in the original entity. (If no data are available, validation can be done by trying to match other existing theories or models or by using similar entities in a different context.)

In many cases, we need to do several simplifications in order to define and solve the equations. For instance, in Example 1.1:

- We used a single-dimensional model, although we have a three-dimensional classroom. Adding a second equation to solve the problem in two dimensions results in extra complexity. In this case, we need to add a new equation, $q_y = -k\,\partial T/\partial y$, and modify the temperature field in the medium $T = T(x, y)$ to be a function of both x and y. Solving this equation is more complex, and this is still a simplification of the three-dimensional problem.
- These equations do not consider transient behavior: what happens if students move? What happens if a window is opened or if we turn the heat on? How is the position of the heating device going to affect the equation definition?
- The equations do not consider combinations of the different possible transients, the materials being used in the room's walls and floors, influence of air flowing in the room, and many other simplifications.
- We need to be able to solve the equations of interest, which can be infeasible or complex (moreover, in many cases just finding the equations to describe the entity under study in detail is impossible).

If we are interested in solving a simple problem like the one in Example 1.1, most of the details in reality can probably be ignored. Nevertheless, if we wanted to use Fourier's equations to study the

heat conductivity of a new material that we want to use for manufacturing, some of the details cannot be ignored (otherwise, the model will lose precision, and the results of the study will be incorrect).

In order to deal with models with a higher complexity, *difference equations* and other *numerical methods* were introduced [3,4]. The idea of these methods is to approximate the equation by discretizing their behavior (which, instead of being continuous and thus computable at every point in time, is now calculated at predefined time steps). The result is not analytical but, rather, numerical, and it will have less precision than the formal model (we cannot obtain solutions for every possible combination of the model's variables). Nevertheless, such methods provide approximate values that are close enough for the problem under study. For instance, in the case of Fourier's equations, we could divide the surface of interest into small elements (assuming a linear temperature distribution along its unit length and a unit area perpendicular to heat flow direction) and obtain the following approximation to the Fourier equation:

$$T_1 = \frac{hT_\infty + \frac{K_1}{L_1} T_o}{h + \frac{K_1}{L_1}} \tag{1.1}$$

Here,

K_1 = the thermal conductivity;
L_1 = the length of the element;
T_∞ = the fluid temperature;
T_0 = the temperature inside the material on a node;
h = the heat transfer coefficient; and
T_1 = the computed surface temperature, which will replace T_0 in the next computation step [2].

The problem-solving activities using these methods can now be described as in Figure 1.3.

The cycle also starts by obtaining experimental data from the entity and then creating equations to model the observed behavior (within the corresponding experimental frames). Nevertheless, as in this case, we cannot find a solution for the equations, so we use a numerical *approximation*. The *computation* is based on a recursive method (that will calculate the values of the state variables at a given time and convert the value as the basis for the next computation). In the past, this task, in general, was carried out by a human expert. The *computation EF* in this case is derived from the model's EF; within the same frame, the computation model should be able to answer the same queries (with a loss of precision due to the approximation). In Figure 1.3 a new step is introduced, which means that we need to carry out an extra check in order to verify the correctness of the results. In this case, we will say that we need to do *verification* of the results obtained by numerical approximation. The idea is that the values we compute need to be checked against the experimental data, while trying to mimic the model's description as accurately as possible.

This approach for problem solving (which was the main technique employed in the last few centuries) made possible a tremendous advance of science and technology. A consequence of such success was the evolution of the associated mathematical techniques, which resulted in the ability to attack new and more complex problems, and new questions to answer (which might require new problem-solving techniques).

Besides the evolution in our learning about nature, the twentieth century witnessed the creation of very elaborate human-made devices (e.g., control systems, intelligent manufacturing, traffic monitoring) that made it difficult (or impossible) to continue using "pencil-and-paper" analysis methods. When we consider such problems (with a few exceptions), they are analytically intractable and numerically impossible to evaluate (unless we simplify the model, which, in most cases, results in solutions that are far from reality).

The advent of computers in the 1940s provided scientists and engineers with alternative methods of analysis. Computers are well suited to deal with approximation techniques, reducing human

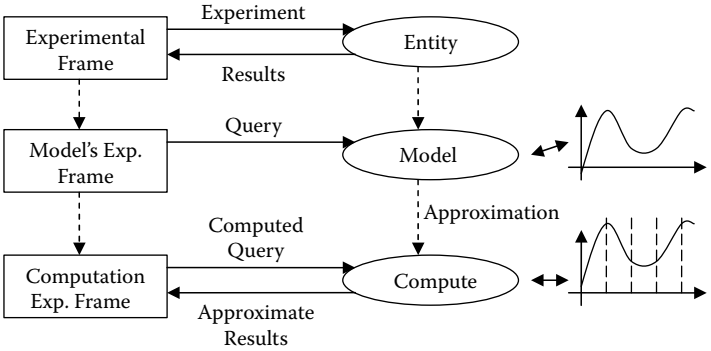

FIGURE 1.3 Problem solving through computation.

computation errors and being able to solve the problems at much higher speed. Thus, since the early days of computing, traditional numerical models were converted into computer-based solutions (and, in many cases, they were called *computer simulations*). The cycle for creating computer simulation (or simply *simulation*) studies can be also represented as in Figure 1.3, with the difference that now the computation model is executed by specialized devices (in the beginning of simulation history, analog computers were used for numerical approximation [5]; today, we use digital computers). The computation EF is now a program that generates test cases for the computational approximation of the model. In this case, the verification activities also need to check the accuracy of the results, while adding an extra step (as limited precision in computers can create erroneous results that can diverge from the expected solutions).

Computer simulation enabled scientists and engineers to experiment easily with "virtual" environments, elevating the analysis of natural and artificial applications to a new level of detail unknown in earlier stages of scientific development and providing great help in the design and analysis of complex applications. Simulated models also can be used for training because they provide cost-effective and risk-free solutions when compared to experimentation. In simulation-based techniques, we find individual solutions for particular problems (as opposed to the general solutions found by analytical methods) using a device (in general, a digital computer) for controlled experimentation and time compression. The constant reduction of the cost of computers (in hand with graphical interfaces, advanced libraries, languages, and other facilities) allowed simulation to become an easy-to-use and flexible technique. Nowadays, modeling and simulation provide a well-developed, well-proven approach to problem solving that advances steadily as more computing power becomes available at less cost.

The advantages of simulation are multiple:

- Decisions can be checked artificially.
- The same model can be reused multiple times.
- Simulations are easier to create and use than many analytical techniques, and they need fewer simplifications.
- The rules used to define the model's behavior can be modified easily.
- During execution of a simulation, we can experiment with varied special cases.
- The user can interact with the simulator, allowing analysis of such interactions.
- Simulation results in shorter design-cycle times and reduced requirements for initial resource investment.
- Simulation provides economic benefits: Research and Development cycles can be improved.
- The original entity is not affected by the study, and it can continue to be used.

In the rest of this chapter, we will discuss some basic ideas about modeling and simulation we introduce in this book. We will present basic concepts of discrete-event dynamic systems, introduce and classify different techniques for modeling such systems, and show how a simulation study is carried out.

1.2 MODELING DISCRETE-EVENT DYNAMIC SYSTEMS

As mentioned in the previous section, computer simulations began by implementing numerical approximations of the differential equations under study with the goal of solving the computation of more complex models in a fast and precise way. Unfortunately, these methods could not be adapted to most of the artificial applications developed during the twentieth century. The Industrial Revolution brought the need to model the behavior of complex apparatuses built by humans (i.e., telephone lines, avionics controllers, automated elevators, etc.), which cannot be adequately described by differential equations or their numerical approximations.

EXAMPLE 1.2

Figure 1.4 shows the observed behavior of a traffic light. Initially, the light is green for 45 s. Then, it switches to yellow for 10 s and finally switches to red for 55 s (after which the cycle is repeated). As we can see, modeling this application with a differential equation is not natural (and it is infeasible to solve in the case of complex combinations including hundreds of traffic lights in a city).

Exercise 1.1

Write a model of the behavior of the traffic light in Figure 1.4 using ordinary differential equations [1, 3].

In order to deal with this kind of problem in a better way, new mathematical theories (in particular, those based on automata theory) were applied in the analysis of these automated devices with discrete components [6,7]. For instance, we can model the behavior of the traffic light in Figure 1.4 with a simple untimed automaton like the one presented in Figure 1.5(a). Such a model can be used to think about correctness of the traffic light behavior (in our example, the cycle from green to yellow to red; if we see a different order when inspecting the model, we will know the model is wrong). We could also build a more complex version like the one in Figure 1.5(b), which uses a timed version of automaton, in which we include the delays for each of the lights. Using this model, we can also think about correctness of the timing properties (and, for instance, detect a wrong duration for a given cycle).

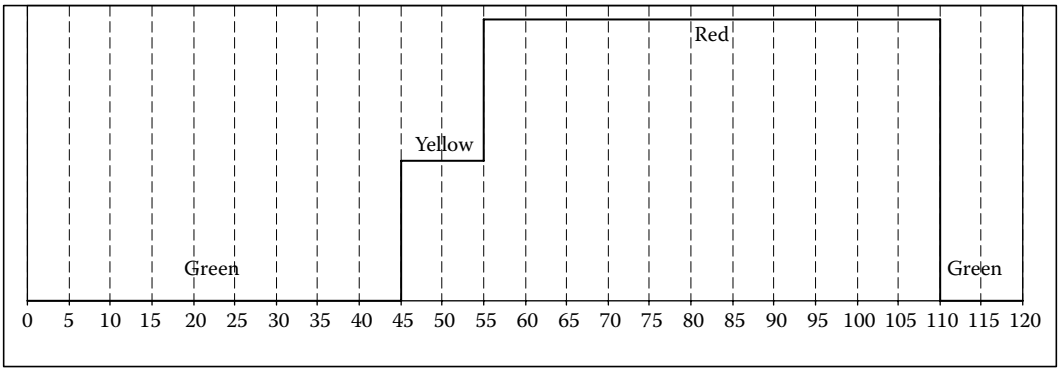

FIGURE 1.4 Observed behavior of a traffic light.

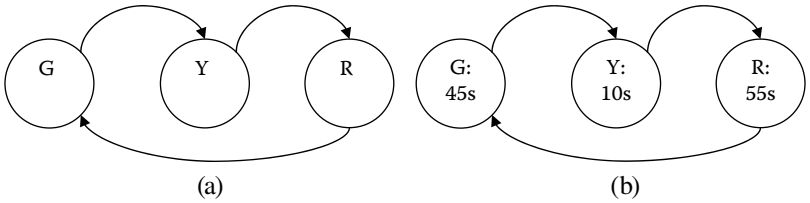

FIGURE 1.5 Simple untimed and timed automata describing the behavior of the traffic light in Figure 1.4.

```
time = 0; State = Green;
Repeat Forever {
      if (State == Green  AND (time mod 110) == 45)   State = Yellow;
      if (State == Yellow AND (time mod 110) == 55)   State = Red;
      if (State == Red    AND (time mod 110) ==110)   State = Green,
      time = time + 5;
}
```

FIGURE 1.6 An implementation of the traffic light model based on the timed automaton of Figure 1.5.

If we now use this model to create a simulator (for instance, an implementation of the timed automata in Figure 1.5(b) using a C-like programming language), we obtain a program in the style of the one introduced in Figure 1.6.

As we can see, we start at time 0 with a green light. The simulator evolves by checking the current state and time and acting according to their values. For instance, after being 45 s in green, the state changes to yellow; after 10 more units, it changes to red and then finally goes back to green. Time is incremented on each cycle using time steps of 5 s (which is the greatest common divisor—g.c.d.—of the three light periods—that is, the minimum value providing enough precision according to the data collected in Figure 1.4).

Exercise 1.2

Build a simulator based on the one in Figure 1.6 and the model in Figure 1.5(b) using a high-level programming language (like Java or C++).

Exercise 1.3

Define a composite automaton describing the behavior of two traffic lights interacting, based on the model in Figure 1.5(a) (in such a way that when one of the automata has a green light, the other is red and vice versa).

Exercise 1.4

Define the behavior of the model introduced in Exercise 1.3 using ordinary differential equations.

Exercise 1.5

Build a simulator to execute the models defined in Exercise 1.3 using a high-level programming language (like Java or C++).

Automata models were defined using discrete values to represent time and the set of states, which works well in many applications but not in others. In order to discuss some of the difficulties, let us consider now a case where the traffic light model introduced in Example 1.2 uses external sensory

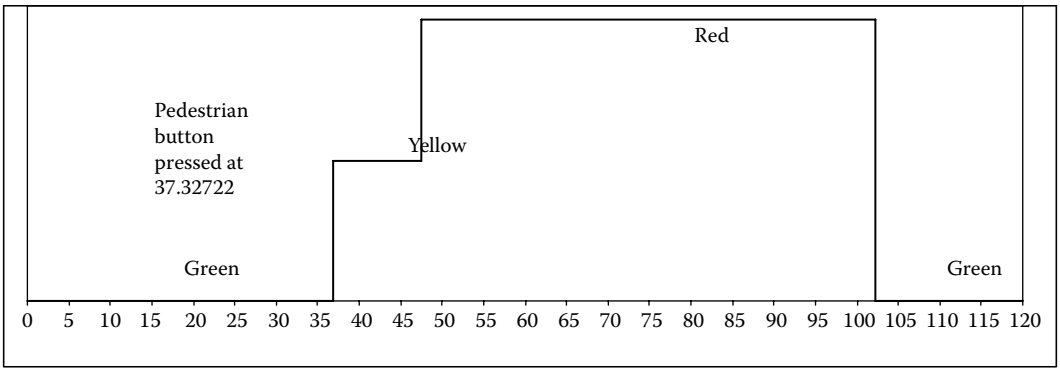

FIGURE 1.7 Observed behavior of a traffic light with a pedestrian button.

information to determine the length of the traffic cycle. We also want to model a button for pedestrians crossing. Figure 1.7 shows an example of the experimental data for such a traffic light.

Here, during the green cycle (which is 45 s long), a pedestrian arrives and presses the pedestrian crossing button, causing a change in the cycle. Such a simple action introduces several problems:

- How do we model the external sensory information in the automaton of Figure 1.5? We want to have cycles that adjust dynamically, so the resulting automaton is much more complex, and it has a larger number of states (if Exercise 1.4 was solved, try now to build the extended automaton for this new adaptation). If we now want to integrate this model for a complex crossing, the number of states might grow exponentially.
- If we need to combine this traffic light with others, how is the variable-timing behavior going to affect the combined automaton?
- When we create a simulator for this model, which would be the right time step to be used? Because neither the arrival time of pedestrians nor traffic flow can be predicted, we cannot find the g.c.d. of every possible cycle (which can be potentially very small or very large). If our requirement is to identify traffic flow changes with high precision, we might use, for instance, a timescale of 0.1 s. Nevertheless, in this case, for every simulation (including those with few or no pedestrians and those with steady traffic), our simple simulator will execute 1,200 cycles (instead of the 22 we used in Figure 1.4). If, instead, we choose a time step of 5 s as before (i.e., to improve the execution performance of the simulator), we cannot accurately model the arrival of vehicles (and in Figure 1.7, we would miss representing the pedestrian button pressed at 37.32722).
- Although this problem does not exist in differential equations (where time is continuous), they are not adequate to represent these kinds of problems (as you could experience if you tried to solve the trivial examples introduced in Exercises 1.1 and 1.3). Instead, automata are easier to understand and better represent these kinds of phenomena.

As we can see, we need different methods to model these human-made applications, in which entities change due to the occurrence of particular *events* and the model's evolution depends on the interactions of such events and their arrival times. For instance, in a computer network, the arrival of a packet is an event of interest, and its arrival time or duration can be of any variable length, depending on the current state of the system and the kind of packet. These kinds of entities, which can be represented with discrete variables and continuous time, are called *discrete-event dynamic systems* (*DEDS*). In DEDS, the states are described by *piecewise constant* trajectories (such as the ones in Figure 1.7). DEDS are naturally concurrent and highly nonlinear; because no transformation

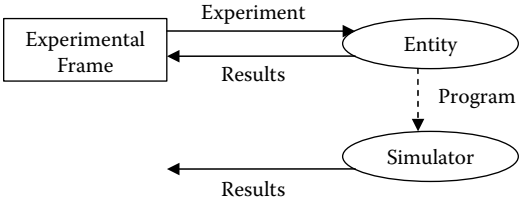

FIGURE 1.8 Building a simulator using a simulation language.

method is available, it is also difficult to find general analytical solutions [8]. The different methods for modeling and simulation (M&S) of DEDS are called *discrete event modeling and simulation*.

Discrete-event M&S assumes that, although time is continuous, only a finite number of events can occur in a given period. Therefore, a discrete-event simulator can be very efficient because we only need to represent the state changes upon occurrence of events. For instance, in order to simulate the traffic light model presented in Figure 1.7, we would only need to execute four cycles (one per color change and one for the button pressed, compared to the 22 or 1,200 previously discussed), while obtaining the maximum precision available to model this traffic light adequately.

Many of the techniques for modeling DEDS only focus on the arrival order of the events, ignoring their timing. They are called *logical DEDS models* and they are used to solve problems as the untimed automata presented in Figure 1.5. Conversely, if timing is a factor, we need *timed DEDS models*.

In order to convert the simulator presented in Figure 1.6 into a discrete-event one, we first need a discrete set of state variables and a clock indicating the current simulation time. Then we use a scheduler to keep a chronological list of events (in general, stored as messages) that represent the state changes. The time in which this will happen, $ti \in \boldsymbol{R}$, is a continuous variable. At every time, the simulator will pick the first event in the list, process it, change the value of the state variables, and cycle to the next event. As we can see, the complexity of the simulation algorithm is higher, and the management of the event list is very important.

In the 1960s, techniques for discrete-event simulation (based on the ideas just discussed) became very popular. In many cases, this resulted in the definition of advanced simulation languages like SLAM, Arena, Simula, or SimScript [9–11]. Although simulation languages can address complex problems, their use lacks the formality of previously existing modeling methods. Using a simulation language helps with problem solving and experimentation, but in most cases their foundation is not rigorous, making the resulting simulation software difficult to test, maintain, and verify. Likewise, changes in the language can produce serious effects in existing models because their semantics are usually not formally defined. Simulation languages do not provide a method abstract enough to think about the problems to solve or to prove properties of the entities under study, which could improve the final quality of the analysis while reducing end costs. Another problem faced by simulation language–based solutions is more subtle and complex to address. The source of many of these problems can be experienced by solving Exercise 1.2 or 1.5, and it is summarized in Figure 1.8: as most simulation languages were not derived from a formal modeling framework, the modeling phase, actually skipped. We start by collecting data from experiments, and we build a piece of software (the simulator), trying to mimic the problem under study (skipping the intermediate modeling phase). Although this method is still useful in many cases, the result is a single-use program approach, which can have several problems (we will use Example 1.2 to discuss some of these problems):

- What happens if we need to reuse the simulation in a different context (e.g., we want to reuse the traffic light controller simulation for a railway controller simulation)?
- How can we reuse the experiments done to test the original model on a different one?

- How do we deal with changes? If we decide to add a blinking green light for left turns (or a blinking red light for a failing controller), we need to modify the software application completely.
- Where is the abstract model we can use to organize our ideas? How do we organize the creation of a new version of the simulator in which we need to study the intersection of six streets with two-way traffic?
- How do we validate the results? What do we do if we find errors in the simulation?

It is difficult to address these issues using the same simulation languages because the model is usually mixed with the experiment and the simulation software. For instance, in Example 1.2, the traffic light controller would be mixed with the generation of pedestrians arriving at the corner and the simulation routines that make time advance and decide what to do next. Even with a simple example like this one, reuse is complex—how can we reuse the control algorithm for the traffic light? Any changes would result in going through the code for the simulation and the experiment because there is no model to use in the verification process. Building a program from the experimental data, as in Figure 1.8, would result in having the original software discarded and a new simulator built from scratch for the next simulation project.

Instead, using a model to organize our ideas can help to create a better product at a reduced cost. Although nonformal models (sometimes called *conceptual models*) can be of help in this task, a formal model like the automata in Figure 1.5 provides much better facilities for verification, reuse, modification and testing. It also provides the basis to define a process that would allow repetition of successful previous experiences, enable reuse, and introduce well-known software engineering practices in the creation of the simulation software.

With these goals in mind, different groups investigated the creation of *formal* discrete-event modeling techniques. Such *formalisms* provide a communication convention written in a mathematical language, which we can use as a guide to provide a nonambiguous specification of the system's semantics [12,13]. Formalisms can be used to represent the entities under study formally, creating an abstract model and providing means for manipulating such abstraction, while being able to translate them into executable models. A *formal model* (i.e., one built using a formalism) provides a sound mechanism to specify the entity under study that can be formally verified. This improves error detection and reduces the development and verification time of the simulation software, enhancing the security and reducing the development costs of the simulation. Going back to Figure 1.3, if we can prove properties of the model before creating its computational version, we can improve the final product at a lower final cost. Simultaneously, a formal mechanism can disambiguate communication, enhancing teamwork by providing a sound notation for the model being constructed.

The rest of this book is focused on one of these discrete-event M&S formalisms, called DEVS (discrete event system specification), a sound M&S theory that has evolved in the last 30 years [12,14–20]. In order to understand the basics of this methodology, we will first discuss some general ideas and we will provide a few general definitions.

1.3 CLASSIFICATIONS OF MODELING TECHNIQUES

Defining taxonomies and classifying methods can help us to have a better understanding of a field under study. Fishwick [13] proposed the following classification, which is based on the types of techniques used (we will use Example 1.1 to show the differences of each of the techniques):

1. *Conceptual modeling:* this technique is based on the creation of an informal *conceptual model* that communicates the basic nature of the process. It provides a vocabulary for the application (which can be ambiguous) and a general description of the entity to be modeled. Example 1.3 shows a simple version of a conceptual model for Example 1.1.

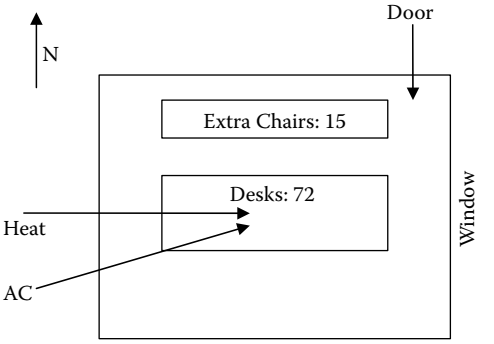

FIGURE 1.9 Classroom organization.

EXAMPLE 1.3: CONCEPTUAL MODEL FOR EXAMPLE 1.1

We are interested in studying temperatures in a classroom throughout the day. The orientation of the building is north (N)/south (S), and there are large glass windows to the east (E). The classroom is 15 × 20 × 3 m. There is a total of 72 movable desks in the room, which are set up (by default) in eight rows of nine desks each. There is also one desk in the front and 15 extra chairs available (without a desk). The classroom has one projector, a whiteboard on the front (which is heading to the S), and a blackboard on the west (W) wall.

There are three teaching seasons (no lectures in summer). Once every hour, an alarm will sound to indicate the end of the lectures. Once every minute, the room's temperature is sensed, and the heating/air conditioning (AC) is activated as needed. In fall and winter, if the temperature goes below 22°C, the heating is turned on, and if gets above 25°C, it is turned off. In spring, if the temperature goes above 24°C, the AC is turned on, and it is turned off when the temperature gets to 23°C (Figure 1.9).

The influence of the temperature outside and in the rest of the building will be neglected. There is a heating/AC source in the middle of the room (on the ceiling) and a door at the back. After extensive experimentation, we know that every time a person arrives in the classroom, the temperature increases 0.05°C after 5 min. Likewise, 5 min after someone leaves, the temperature decreases 0.05°C.

Parameters are room size (15 × 20 × 3 m), desks (72), chairs (87), and board positions (W, S).

State variables are number of students, activity (lecture/no lecture), desk positions (default, circle, semicircle), heating temperature, air conditioning temperature, and season (summer, fall, winter).

Exercise 1.6

Find ambiguities and missing information in this conceptual model. Which of these problems are derived from the expression in natural language? How would you solve these problems?

2. *Declarative modeling:* these techniques are focused on the evolution of the model represented as states (which describe the behavior of the entities under study) and the transitions between them. According to the particular focus of the technique, we can have *state-based* or *event-based* declarative models. *State-based* declarative models, for instance, can be represented as a graph where the vertices represent the entities and the arcs represent the state changes (transitions).

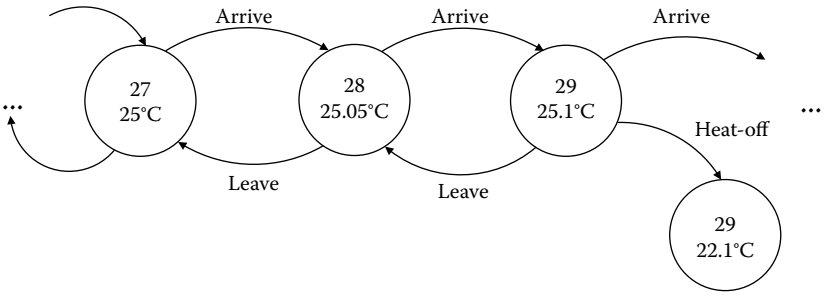

FIGURE 1.10 State-based declarative model of the classroom example.

EXAMPLE 1.4

Figure 1.10 shows a state-based declarative model representing a portion of the system's specification, in which students arrive at or leave the classroom. At a certain point, we have 27 students in the lecture room and a temperature of 25°C. If a new student arrives, we will have a total of 28 students and 25.05°C (this is an untimed model, so timing information is not included). If a student leaves, we return to the previous state; if another student arrives, we have 29 students and temperature will go up (25.1°C). We also show a state representing that the heating has been turned off, which will reduce temperature in the classroom accordingly.

In *event-based* declarative models, we use a graph-based notation with nodes representing events (i.e., the state changes that occur when there is a particular kind of event detected) and links representing the relations between those events. The state changes are associated with each event, and the links can have logical relations associated (defining the relations between events).

EXAMPLE 1.5

Figure 1.11 shows an event-based declarative model for the classroom that represents the same phenomena analyzed in Figure 1.10. If we schedule the arrival of a new student, we will have one more student in the room and temperature will increase. The model can receive more students while there is room available ($q \leq 85$). At any moment, we can schedule a student to leave (in this case, we will decrease temperature and the number of

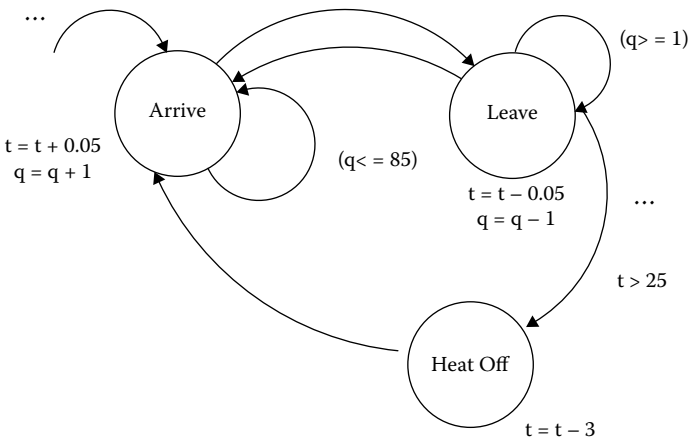

FIGURE 1.11 Event-based declarative model of the classroom example.

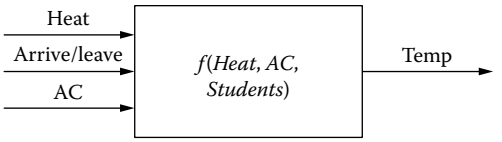

FIGURE 1.12 Functional model of the classroom example.

students in the room). We need at least one student to schedule a student departure ($q \geq 1$). If we schedule the *heat off* event, temperature goes down 3°C.

3. *Functional modeling:* in this case, the model is defined as a "black box" and the input is a signal defined over time. The output depends on an internal function, and the model can use discrete or continuous functions.

EXAMPLE 1.6

Figure 1.12 shows a black box representing the same portion of the system's specification presented in the previous two examples. The function *f* will receive information about students arriving or leaving the classroom. According to the current number of students and level of AC/heat, it will generate the room's temperature (*temp*).

4. *Spatial modeling:* in these techniques, space notions are included (i.e., the relationship between time and space is included in the model).

EXAMPLE 1.7

Figure 1.13 shows a spatial model for the classroom system. At time *t,* the students were distributed in the classroom as in the left part of the figure; at time $t + 1$, we see that a new student has arrived and is now seated in the back row. The number of students will influence the temperature in the classroom.

A different categorization classifies the different modeling techniques according how we represent the state variables and time in the model. Figure 1.14 shows such classification, organizing the various techniques mentioned earlier in this chapter according to their time base and state variables' representations.

As we can see, two criteria are used for the classification:

(a) According to the *time base,* there are **continuous time** paradigms, where time evolves continuously (time is represented a real number), and **discrete time** techniques, where time evolves by advancing in discrete portions (time is an integer number).
(b) According to the *values of the state variables,* there are **continuous** models, where the variables take their values from a continuous set represented as a real number, and **discrete**

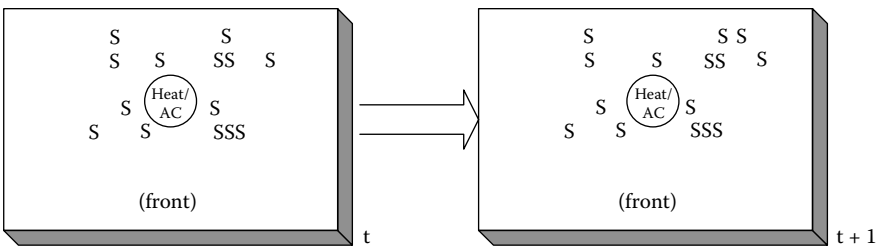

FIGURE 1.13 Spatial model of the classroom example.

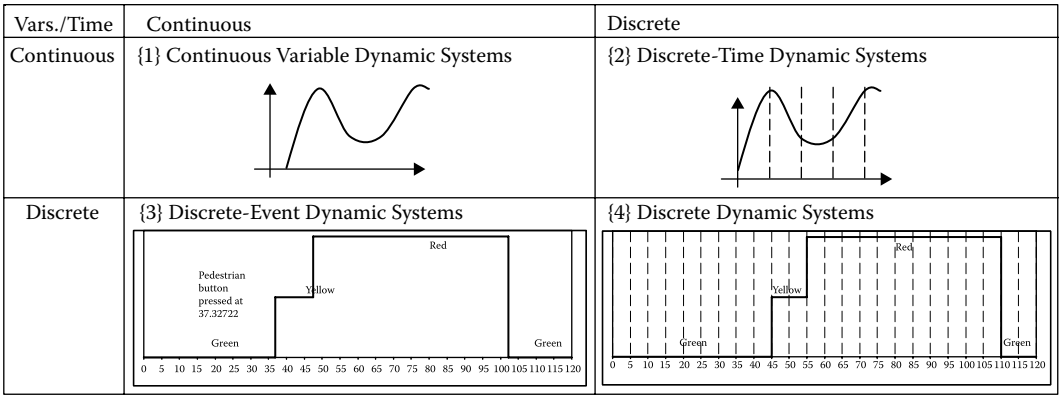

FIGURE 1.14 Classification according the representation of time bases/state variables.

models, where the variables are discrete and can be represented as a finite set of integer numbers.

Figure 1.15 classifies some of the existing modeling techniques according to these criteria.

According to Zeigler, Praehofer, and Kim [17], entities with a behavior like the one depicted in part {1} of Figure 1.14 (which are usually called *continuous variable dynamic systems*) can be defined as *differential equation systems specifications* (*DESS*). If we consider the room's temperature in Example 1.1, we can represent it as a DESS (i.e., temperature in the room can be described as in Figure 1.14{1}). We could use continuous modeling techniques like Bond Graphs, Modelica, and other techniques in Figure 1.15{1} to represent this kind of entity.

Behavior like the one in Figure 1.14{2} (usually called *discrete-time dynamic systems*) can be defined using a *discrete time system specification* (*DTSS*). For instance, the temperature sensor used in Example 1.1 (which measures temperatures at fixed periods) can be represented by difference equations (and other techniques in Figure 1.15{2}).

Figure 1.14{3} shows the behavior of *discrete-event dynamic systems*, which can be described using any of the *discrete event systems specification* (*DEVS*) techniques in Figure 1.15{3} (e.g., the arrival and departure of students in the classroom). Example 1.1 can be modeled with timed FSM, event graphs, etc. as discussed earlier.

Finally, Figure 1.14{4} represents the so-called *discrete dynamic systems*, which can be represented as a specialization of DEVS models in which the events occur at a fixed time. In our classroom, the hourly alarm can be modeled with FSMs, Petri nets (PNs), and other techniques included in Figure 1.15{3}.

Vars./Time	Continuous	Discrete
Continuous	**{1} DESS** Partial Differential Equations Ordinary Differential Equations Bond Graphs Modelica Electrical Circuit Diagrams	**{2} DTSS** Difference Equations Finite Element Method Finite Differences Numerical Methods (Runge-Kutta, Euler, DASSL, and others)
Discrete	**{3} DEVS** DEVS Formalism Timed Petri Nets Timed Finite State Machines Event Graphs	**{4} Automata** Finite State Machines Finite State Automata Petri Nets Boolean Logic Markov Chains

FIGURE 1.15 Classification of modeling techniques according the representation of time bases/state variables.

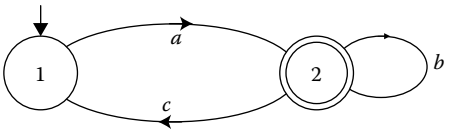

FIGURE 1.16 A simple automaton: $a\{b^*\{cab^*\}^*\}^*$.

1.4 DISCRETE-EVENT MODELING AND SIMULATION METHODOLOGIES

Modeling techniques for DEDS are relatively recent (especially if we compare them with those used for modeling CVDS). In this section, we present a noncomprehensive list of some of the formal modeling techniques created for modeling DEDS (readers interested should consult the references included here).

An *automaton* is defined as a graph representing system states and the transitions between them. The automaton receives a string of symbols as input, and it recognizes/rejects the inputs by advancing through the transitions. The input is read one symbol at a time; depending on the ending state, the automaton will accept or reject the input [6,7].

The automation in Figure 1.16 has an initial state 1 (represented by the arrow) and an ending state 2 (represented by the double circle). If the input *a* is received, the automaton transitions from state 1 to state 2. Once in state 2, it can remain there while receiving the input *b,* it can return to state 1 if it receives input *c,* or it can terminate. Thus, this automaton recognizes the strings {*a, ab, abb, abbb, abbbbb, …, aca, acab, acabb, acabbb, …, abca, abbca, abbbca…*}. This can be represented as $a\{b^*\{cab^*\}^*\}^*$, where * means 0 or more repetitions and {} are used to group strings.

Automata can be deterministic (like the automaton in Figure 1.16), but there are several extensions to the original method, including the nondeterministic automata (in which the transitions can be probabilistic), *input/output* (in which the automaton can trigger a transition when an input is received and can generate outputs, automata noted on the state), etc.

Timed automata, in particular, use *clocks* to describe the model's timing behavior [21]. The automaton is defined as a graph of states associated with clocks that determine the passage of time since the occurrence of an event. Every link is associated with a timing constraint that will define when the transition can be triggered. Whenever a transition executes, the associated clocks are reset. Timing constraints can also be associated with the model states, defining the duration of each of the states.

Figure 1.17 shows the definition of the traffic light model using a timed automaton. The initial state of the model is green, and it will be kept for 45 s. When this time arrives, it changes to yellow, producing an output (the {*yellow*} light). If, before 43 s have passed, a pedestrian presses the crossing button, we switch to yellow (this change will take 2 s). The transitions to red and to green also follow our previous definitions.

Finite state machines (FSMs) can be represented as a graph in which the system's behavior is defined as a finite set of nodes (the model's states) and links between them (transitions between states). A given state reflects the evolution of the model, and transitions are associated with a given logical condition to enable the execution of the transition. When entering a state, an entry action can be executed (and an exit action can be executed when leaving it). Likewise, an input action can be triggered based on the current state and an input [6,7]. An FSM is formally defined as

$$\text{FSM} = \langle \text{S, X, Y, f, g} \rangle \tag{1.2}$$

where

X = finite input set
Y = finite output set
S = finite state set

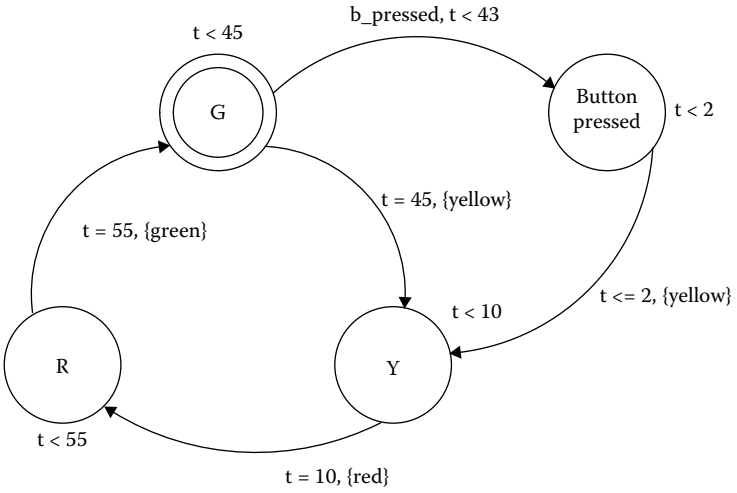

FIGURE 1.17 A timed automaton for traffic light.

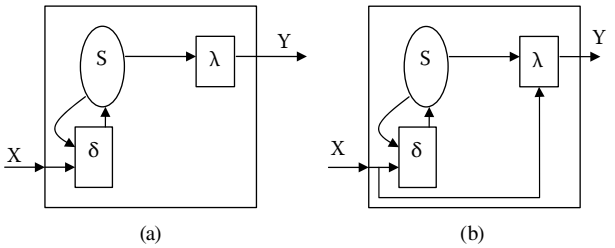

FIGURE 1.18 (a) Moore machine; (b) Mealy machine.

δ = next state function
$\delta = X * S \rightarrow S$
λ = output function
λ: $S \rightarrow Y$: Moore machine
λ: $X * S \rightarrow Y$: Mealy machine

Every FSM is supposed to have an initial state, a next-state function that defines how to obtain the next state in the system, and an output function that uses current state and inputs to generate outputs. According to the relationship between the input sets and output functions, two kinds of FSMs can be defined: Moore machines, in which outputs are independent from the inputs, and Mealy machines, in which, besides the current state variables, the current inputs are analyzed to decide the current output value. These two types are depicted in Figure 1.18.

Timed versions of FSMs associate time with the places or the transitions. Further details about untimed and timed FSMs can be found in Cassandras [6] and Papadimitriou [22].

A *Markov chain* [23] is a discrete-time stochastic model described using a graph (Figure 1.19). Models' states are defined as nodes in the graph, and transitions between states are represented by links. One important property of Markov chains is that they are memoryless; thus, no state has a cause–effect relationship with the previous state. Therefore, knowledge of previous states is irrelevant for predicting the probability of the future states.

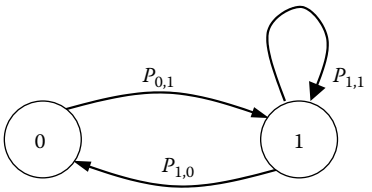

FIGURE 1.19 A simple Markov chain.

In Figure 1.19, we have a binary Markov chain, in which the probability to go from 0 to 1 is $P_{0,1}$, the probability to go back from 1 to 0 is $P_{1,0}$, and the probability of remaining in state 1 is $P_{1,1}$. Markov chains can use discrete time or continuous time, and states can take a discrete value (i.e., values from a finite or a countable infinite set). On a discrete-time chain, we compute the probability to change from one state to the next, and this will happen in discrete time. In continuous-time chains, transitions can occur at any time; thus the transition probabilities must be defined for every instant.

A *Generalized Semi-Markovian Process* (*GSMP*) is a stochastic process (i.e., a collection of random variables over a probability space indexed by time). A GSMP is based on the notion of a state that makes a transition when an event associated with the current state occurs, and the state space is generated by a stochastic timed automaton [24]. Several possible events can compete to trigger the next transition, and each of these events has its own probabilistic distribution for determining the next state.

The GSMP is defined as a set of *locations,* a finite set of *events,* a function that assigns a set of *active* events to each of the locations, a *firing time distribution* associated with each event, and a *transition* function, which takes an active event for a given source locations and returns a set of events and a probability measure. At each transition, new events may be scheduled.

For each event, we set a clock indicating the time when the event is scheduled. It is assumed that the set of scheduled events is uniquely determined by the current state and that there is a unique triggering event for each state transition. A *run* of a GSMP is a sequence of alternating timed and discrete transitions. The run starts at the initial configuration $s_0 = (q_0, e_0)$, where q_0 is the initial location for the GSMP and e_0 is the initial valuation of the events scheduled according to the corresponding firing time distributions (Figure 1.20).

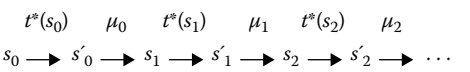

FIGURE 1.20 Execution of a GSMP.

Petri nets [25] define the structure of concurrent systems using a bipartite graph. One type of the graph's nodes, the *places*, represents the system states, and the second kind, the *transitions*, represents the net evolution. The PN example in Figure 1.21 represents a producer/consumer system (the producer is on the left and the consumer on the right of the figure). Here, *L1–L5* are the PN places and *t1–t4* the PN transitions. *L1* represents the producer ready, *L2* a producer that has generated an element to be consumed, *L3* a buffer between the producer and the consumer, *L4* a consumer that is ready, and *L5* a consumer active.

Because a PN is a digraph (i.e., only connections between places to transitions or vice versa are legal), the PN defines a static view of the system's structure (as we can see in Figure 1.21). If we are interested in studying the dynamics of the system, the PN needs to execute. This is done by placing a special mark (called a *token*) on the places (in this case, we have one token on each of places *L1*

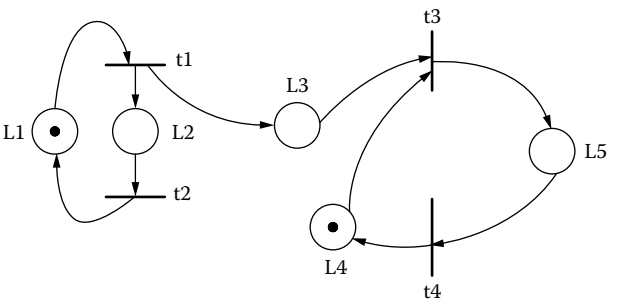

FIGURE 1.21 A producer/consumer Petri net.

and *L4,* representing that the producer and consumer are ready to start). The number of tokens per place is unlimited, and the number of tokens in the whole net defines the net's *marking.* The PN evolves by executing the transitions, which is done by taking one token per *input* place (i.e., those with a link from a place to the executing transition—for instance, *L1* → *t1*) and placing one token on every *output* place (i.e., those with a link from the executing transition, i.e., *t1* → *L2, L3*). Each execution of a transition is called *firing* the transition. We can only fire a transition that is *enabled* (i.e., one having at least one token on each input place). The execution of a transition is atomic, and if more than one transition is enabled, they execute in nondeterministic fashion.

Figure 1.22 shows the execution of the PN in Figure 1.21, using a *reachability tree*, which contains one node per marking and one link per firing. For instance, the initial marking in Figure 1.21 can be represented as the configuration: (1,0,0,1,0). With this marking, only *t1* is enabled (it is the only transition with a sufficient number of tokens in the input places). If we fire *t1*, we obtain the marking (0,1,1,1,0), which enables *t2* and *t3*. Any of them can be fired (and which one is fired is decided in nondeterministic fashion). For instance, if we fire *t2*, we obtain (1,0,1,1,0), which is a similar marking to the root of the tree but with an extra token in *L3.* At this point, if we repeat the firing sequence *t1* → *t2*, we will obtain the same marking, with one extra token in *L3.* Thus, instead of expanding this branch of the tree indefinitely, we put an asterisk in the corresponding position for *L3* (meaning that the number of tokens in this place can grow indefinitely) and stop expanding this branch of the tree.

If, instead, we fire *t3*, we obtain (0,1,0,0,1). At this point, we can fire *t2* or *t4,* etc. When we cannot fire further transitions or we obtain a marking repeated in the tree, we stop expanding the tree. For instance, after firing *t1* we obtain (0,1,0,0,1), which is repeated in the tree; thus, we stop expanding this branch.

By studying the tree, we can study concurrency problems, deadlocks (i.e., a PN that does not evolve and have resources in a cycle), unbounded buffering problems, starvation (transitions that can never execute), etc.

Exercise 1.7

Study the reachability tree in Figure 1.22 and discuss the meaning of each of the states.

Exercise 1.8

Add two extra tokens in *L1* and rebuild the reachability tree.

Exercise 1.9

Using the reachability tree in Figure 1.22, carefully check whether the occurrence of concurrent states is correct. Find cases of deadlock, starvation, or any other problems (if there are any). Repeat the exercise for the tree obtained in Exercise 1.8.

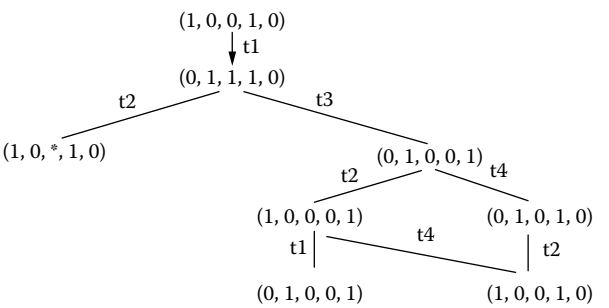

FIGURE 1.22 Reachability tree for Figure 1.21.

Exercise 1.10

Delete the token in *L4*, move it to *L3*, and repeat Exercises 1.8 and 1.9.

Queuing networks are based on a customer–server paradigm, in which customers make service requests to the servers and these requests are queued at the server until they can be serviced. The arrival time for customers and the service time at a server are described as stochastic models. By defining the number of servers and the buffering capacity on each of them, we can determine performance metrics (including the number of customers in line, throughput—number of customers serviced per time unit, turnaround times, etc.). Different policies can be used (priorities; preemption; first in, first out [FIFO]; etc.). Traditional queuing networks model only a single layer of customer–server relationships. Layered queuing networks (LQNs) allow for an arbitrary number of client–server levels [26,27]. LQNs can model intermediate software servers, and they can be used to detect software deadlocks and software as well as hardware performance bottlenecks. The layered aspect of LQNs makes them very suitable for evaluating the performance of distributed systems [28].

The formal language of *calculus of communicating systems* (*CCS*) provides primitives for concurrency and parallelism, based on synchronous communications between exactly two components. The language expressions are interpreted as a labeled transition system, and bisimulation can be used to prove equivalence of models [29].

Temporal logic is a system of rules and symbols used for representing propositions that can include the timing properties of the system [30]. It consists of a logic set of propositions that view time as a sequence of states and that can be true or false according to their state and their time of occurrence. Temporal logic has been used to verify formally timed automata. The idea is to check predictability of certain conditions according to the time that they occur, conditions that might eventually arise, or others that are guaranteed not to occur.

Communicating sequential processes (CSP) is a formal language based on process algebra that has been widely used to model concurrent systems [31]. Models are described using independent processes that interact with each other through message-passing representing the occurrence of events. The processes' primitive elements include:

- Prefixing ($\rightarrow$) connects an event with a process. For instance, if we write $x \rightarrow P$, that means that the process P will wait for the event x, after which P will execute.
- Inputs (?) represent the reception of inputs for the process. For instance, when we write $?x{:}X \rightarrow P(x)$, this represents a process P that, upon reception of x (which belongs to the set of inputs X), executes $P(x)$.
- Choice: the *external* choice permits us to select one of several actions. For instance, $P = x \rightarrow P1 \,\square\, y \rightarrow P2$ can execute either of the two processes, depending on the event received (x will trigger *P1* and y will trigger *P2*; if both occur, the choice is nondeterministic). The *internal* choice picks one of them in a nondeterministic fashion. For instance, $P = x \rightarrow P1 * y \rightarrow P2$ can choose either of the two processes, independently from their inputs. Other operators include a choice with conditionals, Boolean operators, etc.
- Recursion is defined by having the identifier of the process at both ends of the equation—for instance, $Clock = tick \rightarrow tack \rightarrow Clock$, which will repeat the events {*tick, tack*} forever.
- Parallel: if we have $P1\ X\|Y\ P2$, *P1* and *P2* execute in parallel and they synchronize using the intersection of X and Y.

State transition diagrams usually suffer from a "state explosion," in which the number of states and transitions becomes unmanageable. *State charts* [32] introduce hierarchy into the model, allowing definition of varied behavior at different levels, as seen in Figure 1.23. A state is represented as a rectangle with round corners, which can be subdivided into a list of internal transitions (which are performed while the element is in the state). The start state is represented as a circle and the final

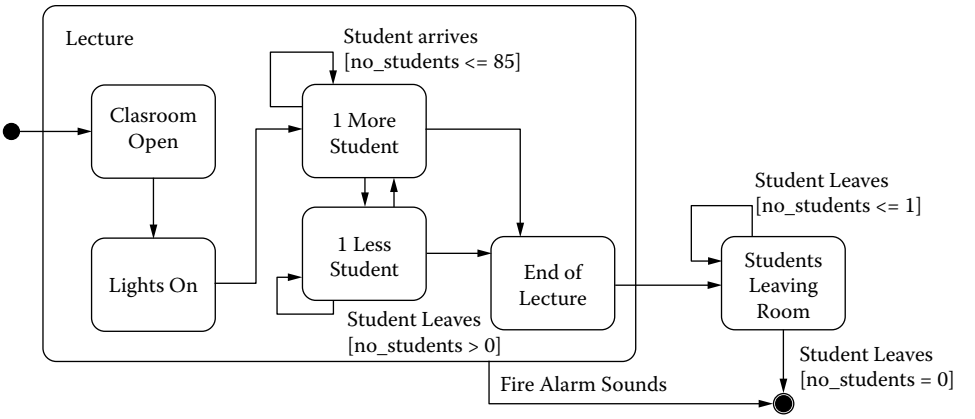

FIGURE 1.23 State chart example.

state as a double circle. The transitions between elements are represented with arrows. Numerous other constructors are available (concurrent separators, complex transitions, event messages, etc.).

In Figure 1.23, we have two states: a lecture and an end of lecture (where students leave the room). As we can see, we can hierarchically subdivide the lecture state into substates that will represent the start of the lecture, arrival and departure of students, and the end of the lecture.

State charts are also used in the UML dynamic model to show the behavior of a single class [33]. State charts introduce the concept of modeling of concurrent machines and superstates (i.e., a state that contains other states), thus making it possible to have two or more concurrent states (as opposed to classic state machines, which can only be in one state at a time). States and superstates can be nested, each with its own initial and final states.

Specification and Description Language (*SDL*) was created to specify in a nonambiguous way the behavior of real-time applications. It was originally focused on communication systems, by providing a graphical and textual representation with equivalent semantics. A system is defined as a set of extended FSMs that can be interconnected [34].

Event graphs are oriented graphs that represent the organization of the events of a discrete event system [35]. As seen earlier in Figure 1.11, events constitute nodes of the graph; that is, the vertices represent the state transition functions, and the links between nodes capture the scheduling of such events. Each link starts at the node performing the scheduling operation (which represents an event), and it ends at the node representing the event to be scheduled. Each scheduling relationship has an associated delay and condition (a Boolean function of the state), and an event is scheduled only when the condition is true.

Systems-theoretical approaches derive from systems theory [36]. Systems theory represents every entity under study using the concept of *system*, which is seen as a collection of objects and their interactions. In systems theory, the system's global behavior is seen as a composition of the individual behavior of the components, and we can find emergent behavior that is not explicitly defined in the parts of the system. Systems theory is based on the idea that every phenomenon can be viewed as a mathematical relationship among a set of entities in the system. The theory is generic and tries to find common behavior and properties in different fields of study (for instance, hydraulics, economy, biology, or social sciences), thus providing a unified view of science and engineering.

Systems theory distinguishes the structure (internal constitution) of a system and its behavior (external manifestation). Each component is seen as having inputs, outputs, and internal structure that dictates how inputs and states are transformed into outputs. If the model only allows observing the external manifestation, we call it a **black box** (or **behavioral**) model; when we analyze it, we only consider the input/output trajectories observed. These models contain a representation of observable and unobservable variables, and the system's behavior is described as a set of input/

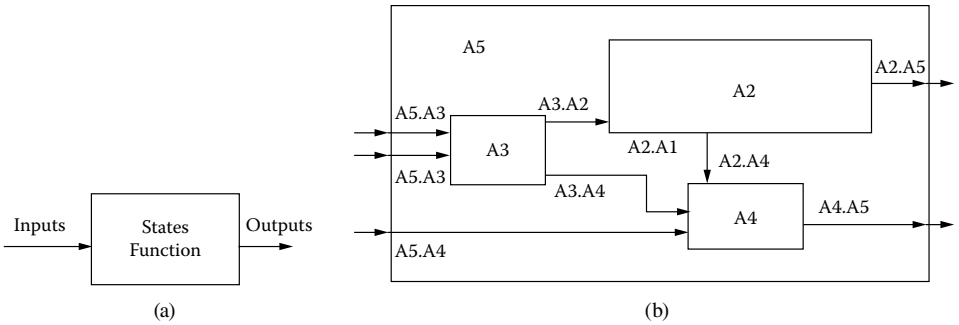

FIGURE 1.24 (a) Behavioral model; (b) structural model.

output data that we can observe about the system of interest. When the model represents the internal constitution of the system, we call it a **white box** (or **structural**) model. Structural models introduce concepts of *decomposition* and *coupling,* and they refer to the component elements in the model (as opposed to the behavior they generate). Decomposition focuses on how to divide a system into components, and coupling focuses on how to combine the components to reconstruct it. Using these concepts, we must explicitly define the components of the real system, using decomposition and coupling concepts (Figure 1.24).

In the case of Example 1.1, we can build a behavioral model of the temperature of the whole room by defining a transition *function* that will be activated every time a student arrives/leaves, updating the temperature and number of students in the room (*states*). On the other hand, a structural model would consider the different heat/cold sources, the student's seating area, and the connection to the rest of the building. In this case, we are not interested in the behavior but, rather, in the decomposition of the model into subcomponents and their interrelation.

Systems theory also focuses on different views of the same system. For a given system, we can have many models, which will be valid with respect to the objective for which it was created (that is, it is going to be able to answer to queries according to the objective and the experimental frame under which it was created). The bottom line is that a *complete* model never exists because there is an indefinite number of models that can be created for the same system. Simultaneously, every model can have many different implementations (simulations), and a variety of experiments can be carried out. In Example 1.1, we can use the source system and create an experimental frame to study the temperature (as described in the different examples we have discussed). The same source system could also be used for studying weight of the student population, its gender, distribution of students in classrooms, scheduling of lectures, construction of the building based on occupancy, evacuation under emergency, etc.

Discrete-Event Systems Specifications (DESS) formalism [12,15–17,18] is a mathematical modeling technique derived from systems theorythat allows the construction of hierarchical and modular models, providing a well-defined coupling of components. Given its hierarchical nature, DEVS allows the coupling of existing models modularly, allowing us to build complex systems from simple ones. DEVS theory provides a rigorous methodology for representing models, and it presents an abstract way of thinking about the world independently of the simulation mechanisms.

The formalism is based on general system theory, and it allows us to represent all the systems whose input/output behavior can be described by sequences of events. The generality of DEVS is derived from the fact that it permits the modeling of systems with a set of infinite possible states where the new state after an event arrival may depend on the (continuous) time elapsed in the previous state.

Exercise 1.11

Classify each of the previous techniques according to Fishwick's taxonomy.

1.5 SOME DEFINITIONS

Having discussed the basic ideas we will cover in the rest of the book, we are now ready to introduce some definitions.

According to systems theory, a *system* is a natural or artificial entity, real or abstract, that is a part of a given reality constrained by an environment. It can be seen as an ordered set of related objects that evolve through different activities, interacting to achieve a goal. It is also called the *real system* (or the *system of interest*), and it is seen as a source of observational data, which is viewed through an *experimental frame* (EF) of interest to the modeler [17]. The system's observational data are used to define the model's *behavior*, which is defined as a specific form of data observable in a system over time within an experimental frame.

A *model* is an understandable representation (abstract and consistent) of a given system that we use to understand it better. Models can be built in a variety of ways, and they have different meanings according to the individual doing the modeling. For architects, a model can be the drawing of a floor map or a maquette; for a biochemist, a model can be a three-dimensional representation of a molecule; etc. In this book, we are not interested in such static models. Instead, we are interested in models that have *dynamic behavior* (i.e., models that exhibit time-varying changes), we want to study how the model organizes itself over time in response to imposed conditions and stimuli. The objective is to predict how the system will react to external inputs or structural changes. The model's behavior is generated using specific rules, equations, or a modeling formalism with the goal of generating behavior that should be indistinguishable from the one of the system within one or more models' EFs. The EF defines the conditions under which a system or a model is observed or experimented with; thus, problem solving is related to the EF within which the model is analyzed [17].

The process of thinking and reasoning about a system in order to abstract the description of the model from reality is called systems *modeling.* We will use the term *paradigm* to refer to the concepts, laws, and mechanisms that are used to define a set of models. It is important to keep a clear separation between the systems of interest and the models that we use to think about them. (A model is an abstract representation of the system rather than the system itself, although it is easy to confuse them because we are used to thinking about models for reasoning about real systems.)

Discrete-event modeling is based on the notion of *event,* which is defined as a change in the state of the model. An event occurs at a given *instant* (called the *event time*) and causes the model to *activate* in order to produce a *state change* (e.g., at least one attribute in the model will change). Finally, a model's *state* is the set of values of all the attributes of the model at a given instant. The model's attributes are usually stored in variables; *state variables* are those that will influence the evolution of the model's behavior [37].

A model is an *abstract* representation of the system of interest because the model is an aggregated elaboration of the information provided by the system (with a format based on rules, equations, and relations between components). In modeling, we define *abstraction* as the basic process used to extract a set of entities and relations from a complex reality [17]. During this abstraction process, information is lost; however, a higher level of abstraction allows us to better define the model's behavior and to prove properties of the system by manipulating the abstract model definition while addressing the problem at the right level of complexity. Figure 1.25 shows an example of a variety of abstraction levels in a real system.

In Figure 1.25, we start with a continuous signal in the plant that is read by sensors. In order to model the behavior at this level of abstraction, we need to use a continuous system methodology (for instance, bond graphs). If we want to feed information from the plant to a digital computer, we sample the input data (and in order to model this sampled data we need to use a discrete time technique). The sampled data information from the plant is stored in an image of the plant, and we use intelligent agents within a supervisory control system to process the inputs and react accordingly (for instance, to activate alarms under emergency). The intelligent agents need to identify

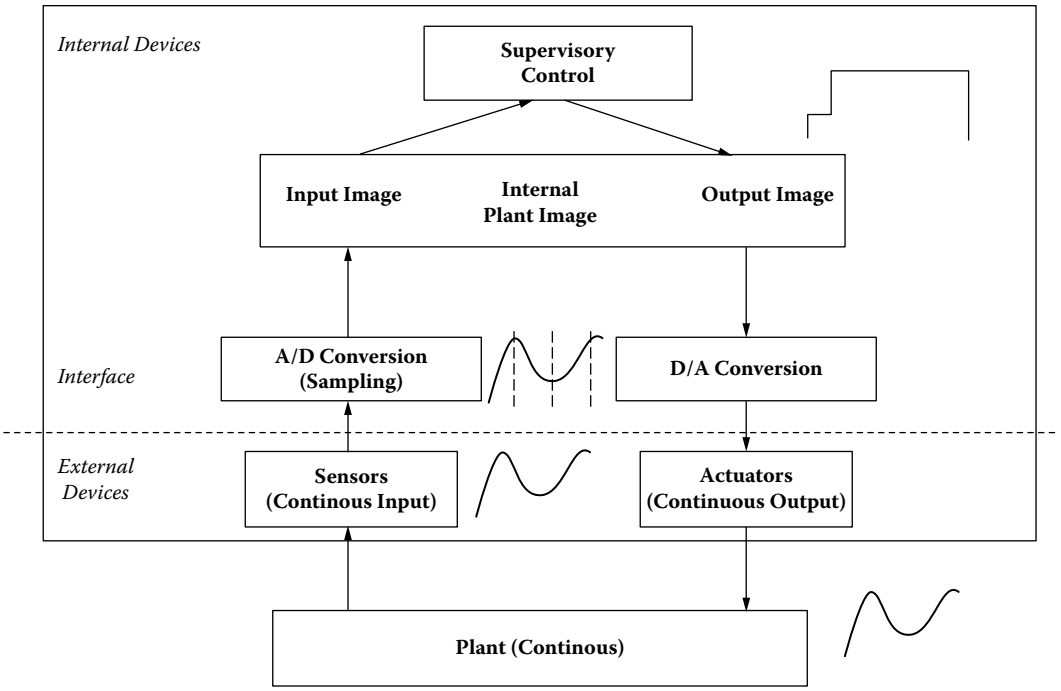

FIGURE 1.25 Different abstraction levels in a plant model.

meaningful events to react to; thus, a discrete-event model, detecting important events to be processed by the supervisory system, would be the best choice. As is clear in this process, we lose information at each step, but we are able to make decisions that are more precise at each level by using the right level of abstraction. Current systems of interest usually include components defined with multiple techniques (also referred to as *multimodeling*). Likewise, each of these models might have different levels of *resolution* according to the needs. (For instance, under emergency, we might downgrade the sampling rate to better utilize the computer resources in dealing with the emergency, thus reducing the quality of the input data while the emergency lasts.) These *multiresolution* models are needed more and more frequently as they provide multiple versions of a model that can be easily adapted and reused.

We can now define *simulation* as the reproduction of the dynamic behavior of a system of interest with the goal of obtaining conclusions that can be applied to the system.

A *simulation study* refers to a set of simulation-based experiments. Typically, the model associated with a simulation study is implemented as a computer program (also called a *simulation model* or a *simulator*). The *simulation experimental frame* contains information about the experimental conditions, parameter values, and behavior generation mechanisms. A simulator may interface with real-world entities (e.g., a moving platform for a driving simulator) or with humans (e.g., a driver in the simulator). We call the first a simulator with *hardware in the loop* and the second a simulator with *human in the loop.*

In order to be able to create a model and a simulator that represent the system with precision, we need to carry out verification and validation (V&V) activities. We use the term *validation* to refer to the relationship of the model, the system of interest, and the EF. A model's validation answers the question of whether it is impossible to distinguish the behavior of the model and the system within the EF. *Verification* is the process of verifying that a simulator of a model correctly generates the desired behavior [17].

EXAMPLE 1.8

Let us use Example 1.1 to explain some of the definitions just presented. The *experimental frame* consists of different classroom configurations for the study (students entering/leaving and varied heating/cooling levels). We created different models: a *spatial* one in Figure 1.13 and a *declarative* one (finite state machine) in Figure 1.10. A student arriving in the classroom is an *event* of interest for this model. Two *state variables* will change upon occurrence of such event: the number of students in the room and the room's temperature (the composite change of these two attributes is the room's *state change*). This event will occur at the *instant* when the student enters the room (represented by a real number). The FSM and the spatial models introduced in previous figures are *abstract* representations of the actual system (for instance, the FSM in Figure 1.10 does not contain any information about the student's age or gender, which are attributes of the real system that we are not interested in modeling within our EF). A *simulator* for this model would implement the FSM in software, and a simulator's EF would generate a variety of independent tests to run the program. A different simulator would implement a software version of the spatial model in Figure 1.13. In order to *verify* the simulator, we need to check that the trajectories generated in runtime coincide with those defined by the model (for instance, if a new student arrival is simulated, we see the number of students increasing, and the temperature increasing, as described in the model). Finally, *validation* activities should ensure that what we see in the model and in the simulation results coincides with reality. (For instance, if we see temperatures decreasing when a new student arrives, we know that this simulated behavior is not valid; we have first to check the model and, if its specification is erroneous, we need to fix it. Otherwise, we need to modify the model's simulator.)

Once the study has been finished and the results are obtained, we might need to use them on the original system of interest. According to the objectives for the study and the decisions to be taken on the system of interest once it has finished, we can consider the following kinds of simulation models:

- *Exploration* models are models that are used to understand the operation of the system better.
- *Prediction* models are models that are used to predict the future behavior of the system.
- *Improvement* models are models that are used to optimize the performance of the system through the analysis of different alternatives.
- *Conception* models are models that are used when the system does not exist yet, and the model is used to test different options prior to construction.
- *Engineering design* models are models that are used to design devices in engineering applications (ranging from bridges to electron devices). Simulation permits exploring different design options and helps to choose the best.
- *Rapid prototyping* models are models that permit obtaining quickly a working model that can be used to test ideas and get early feedback from stakeholders.
- *Planning* models are models that provide a risk-free mechanism for thinking about the future in different fields of application (ranging from manufacturing to governance).
- *Acquisition* models are models that involve choosing proper equipment. Very large pieces of equipment (e.g., helicopters, airplanes, submarines) are extremely expensive. M&S can help us to decide during the purchasing process, enabling the customer to explore different alternatives without the need of constructing the equipment prior to taking the decision.
- *Proof of concept* models are models that can be used to test ideas and put them to work before creating the actual application.
- *Training* can be used to provide controlled experiments to enhance decision-making abilities and teach fundamental concepts for defense applications (also called *constructive* simulation). Other training examples include *business gaming* and *virtual* simulators (which are usually human-in-the-loop simulators to learn and enhance motor skills when operating complex vehicles).

- *Education* models are models that can be used in different sciences to provide insight into the nature of dynamic phenomena as well as the underlying mechanisms.
- *Entertainment* models are models that involve games and animation, the two most popular applications of simulation.

1.6 PHASES IN A SIMULATION STUDY

There have been different kinds of life cycles proposed for studies in modeling and simulation [17,38–41]. In this section, we summarize the basic steps that should be considered in doing a simulation study. The life cycle does not have to be interpreted as strictly sequential; it is iterative by nature, and sometimes transitions in opposite directions can appear. Likewise, some of the steps can be skipped, according to the complexity of the application. It is highly recommended to use a spiral cycle with incremental development for steps 2–8, which can cause a revision to earlier phases. Each phase in the spiral cycle should end with a working prototype including more functionality than the previous cycle:

1. *Problem formulation:* the simulation process begins with a practical problem that requires solving or understanding. It might be the case of a cargo company trying to develop a new strategy for truck dispatching or an astronomer trying to understand how a nebula is formed. At this stage, we must understand the behavior of the system of interest (which can be a natural or artificial system, existing or not), organizing the system's operation as objects and activities within the experimental framework of interest. Then we need to analyze different alternatives of solutions by investigating other previously existing results for similar problems. The most acceptable solution should be chosen (omitting this stage could cause the selection of an expensive or wrong solution). We also must identify the input/output variables and classify them into **decision** variables (controllable) or **parameters** (noncontrollable). If the problem involves performance analysis, this is the point at which we can also define performance **metrics** (based on the output variables) and an **objective function** (i.e., a combination of some of the metrics). At this stage, we can also do risk analysis and decide whether to follow or discard the project.
2. The *conceptual model* must be defined. This step consists of building a high-level description of the structure and behavior of the system and identifying all the objects with their attributes and interfaces. We also must define what the state variables are, how they are related, and which ones are important for the study. In this step, key aspects of the requirements are expressed (if possible, using a formalism, which introduces a higher degree of precision). During the definition of the conceptual model, we need to reveal features that are of critical significance (e.g., possibility of instability, deadlock, or starvation). We must also document nonfunctional information—for instance, possible future changes, nonintuitive (or nonformal) behavior, and the relation with the environment.
3. In the *collection and analysis of input/output data* phase, we must study the system to obtain input/output data. To do so, we must observe and collect the attributes chosen in the previous phase. When the system entities are studied, we try to associate them with a timing value. Another important issue during this phase is the selection of a sample size that is statistically valid and a data format that can be processed with a computer. Finally, we must decide which attributes are stochastic and which are deterministic. In some cases, there are no data sources to collect (for instance, for nonexisting systems). In those cases, we need to try to obtain data sets from similar systems (if available). Another option is to use a stochastic approach to provide the data needed through random number generation.
4. In the *modeling* phase, we must build a detailed representation of the system based on the conceptual model and the I/O data collected. The model is built by defining objects, attributes, and methods using a chosen paradigm. At this point, a specification model is

created, including the set of equations defining its behavior and structure. After finishing this definition, we must try to build a preliminary structure of the model (possibly relating the system variables and performance metrics), carefully describing any assumptions and simplifications and collecting them into the model's EF.

5. During the *simulation* stage, we must choose a mechanism to implement the model (in most cases using a computer and adequate programming languages and tools), and a simulation model is constructed. During this step, it might be necessary to define simulation algorithms and to translate them into a computer program. In this phase, we also must build a model of the EF for the simulation.
6. *V&V:* During the previous steps, three different models are built: the conceptual model (specification), the system's model (design), and the simulation model (executable program). We need to verify and validate these models. Verification is related to the internal consistency among the three models (is the model correctly implemented?). Validation is focused on the correspondence between model and reality: are the simulation results consistent with the system being analyzed? Did we build the right model? Based on the results obtained during this phase, the model and its implementation might need refinement. As we will discuss in the next section, the V&V process does not constitute a particular phase of the life cycle, but it is an integral part of it. This process must be formal and must be documented correctly because later versions of the model will require another round of V&V, which is, in fact, one of the most expensive phases in the cycle.
7. *Experimentation:* We must execute the simulation model, following the goals stated in the conceptual model. During this phase, we must evaluate the outputs of the simulator, using statistical correlation to determine a precision level for the performance metrics. This phase starts with the design of the experiments, using different techniques. Some of these techniques include sensitivity analysis, optimization, variance reduction (to optimize the results from a statistical point of view), and ranking and selection (comparison with alternative systems).
8. In the *output analysis* phase, the simulation outputs are analyzed in order to understand the system behavior. These outputs are used to obtain responses about the behavior of the original system. At this stage, visualization tools can be used to help with the process. The goal of visualization is to provide a deeper understanding of the real systems being investigated and to help in exploring the large set of numerical data produced by the simulation.

The rest of this book will concentrate mostly on phases 4, 5, and 6, using the DEVS formalism as the chosen paradigm. We will explain how to do advanced visualization for phase 8, and we will show how to conduct some experiments (phase 8). The focus here is on the practitioner's point of view, and those interested in understanding the details of the underlying theories should consult references 12, 17, 38, 39, and 41. Input/output analysis is exhaustively covered in references 6, 39, 42–44, and others. Likewise, we will not cover random number generation because it is covered in the previous references. We will focus on how to create an advanced simulation toolkit based on DEVS modeling and simulation methodology, concentrating on how to reduce the development costs of a simulation (by merging phases 2 and 4 and reducing phase 5) while providing good facilities for incremental development, V&V, experimentation, and maintenance.

1.7 VERIFICATION AND VALIDATION (V&V)

The credibility of the results of the simulation depends not only on the correctness of the model, but also on how accurate the formulation of the problem is expected to be [45]. Thus, we must use various V&V techniques throughout the life cycle of the simulation study. As discussed earlier, we call *validation* the process of determining that a model is a correct representation of the real world and that its behavior corresponds to the requirements of the model (i.e., validation is related to the

correct formulation of the model). We say a model is *valid* when it is impossible to distinguish system and model within the experimental frame. We can recognize different types of validity [17]:

- *Replicative validity:* For every possible experiment within the experimental frame, trajectories of model and system agree within acceptable tolerance.
- *Predictive validity:* Within an experimental frame, it is possible to initialize the model to a state such that, for the same input trajectory, model output trajectory predicts system output trajectory within acceptable tolerance.
- *Structural validity:* The model is able to replicate the data observed from the system but also mimics step by step and component by component the way in which the system does its transitions.

Verification, on the other hand, is the process of checking that a simulator of a model correctly generates its behavior according to the model's specification. A *correct simulation* faithfully generates model behavior in every simulation run.

In large organizations (mostly related to M&S in the defense industry), an *accreditation* phase can be required. This is the official process of gaining approval for model use, which certifies that the model satisfies the specifications. *Specific* accreditation includes the model and its environment (input data, aptitude of the personnel, performance of the simulator, etc.).

The result of the activities of V&V must not be considered in absolute form (e.g., right/wrong). A simulation model is constructed with certain objectives described in the conceptual model and the experimental frame. As in any software project, it is recommended that V&V should be executed by an independent team to prevent biased decisions [45]. Likewise, the software cannot be tested completely, and standard testing techniques should be used, trying to cover the largest percentage of cases within the domain [46]. The objective is to increase confidence in the model, based on the study of the objectives. Likewise, the satisfactory test of each submodel (unit test) does not imply the correctness of the whole (integration test).

Following the life cycle described in Figure 1.26, we need to carry out the following activities:

1. Verification of the conceptual model and of the problem under analysis: We must determine that the problem we are solving corresponds to the real system. It is essential to understand the requirements of the system of interest, justifying all the assumptions made during the definition of the conceptual model. We have to check that such assumptions are appropriate and that the conceptual model represents the real system based on the proposed objectives.
2. Verification of the design: The purpose of design verification is to ensure that the specifications reflect the conceptual model accurately.
3. Validation of the system's model: We must verify that the model corresponds to the system of interest within the EF. We must also verify that the model is represented with a sufficient degree of accuracy. As part of this process, we must ensure the quality of the data used in the different phases of the model.
4. Verification of the simulator: We must ensure that the software implementation of the model reflects its specification. The model's behavior must be tested to cover the maximum percentage of cases possible.
5. Validation of the experimental results: The credibility of the model is the result of having completed each of the different activities of V&V mentioned earlier satisfactorily and ensuring that the results given by the simulator are compatible with those of the real system.

A variety of V&V techniques can be employed (discussed in detail in references 40, 43, and 45–47), which can be categorized as the following:

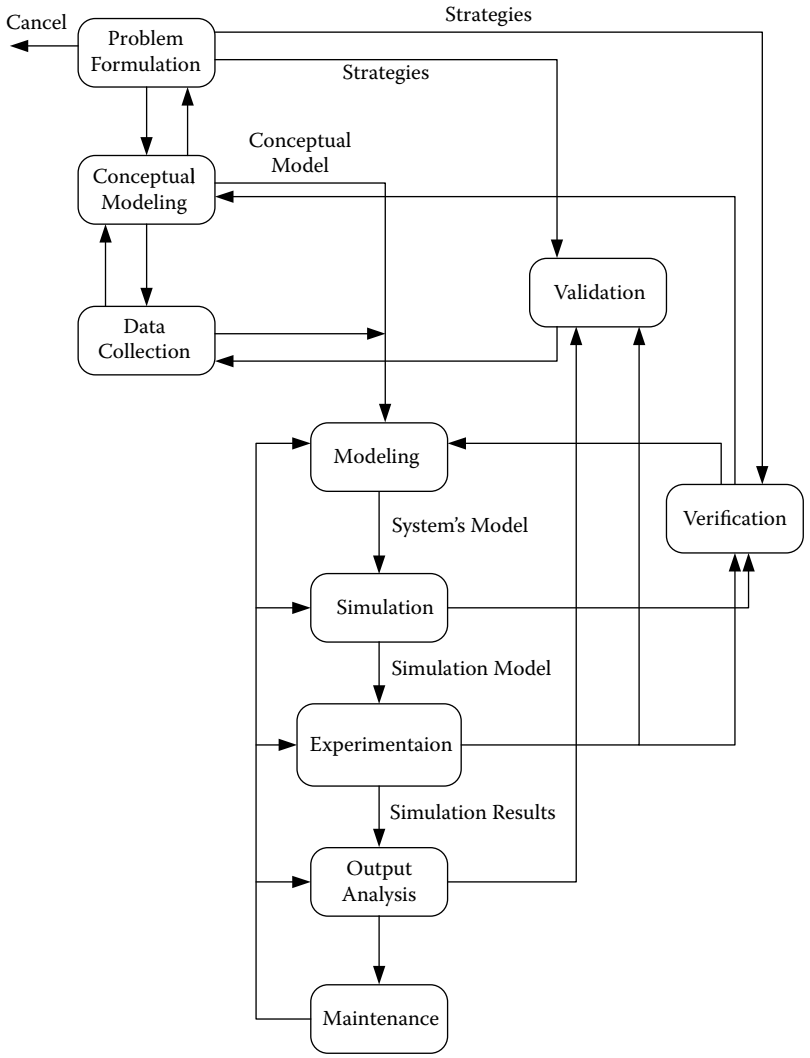

FIGURE 1.26 Steps in M&S studies.

- *Informal* techniques lack mathematical foundations and are based mainly on human reasoning. These are the most common techniques, including inspections, Turing test (i.e., making an expert interact with the simulator, trying to make sure the expert cannot recognize the differences between the simulator and the real system), and animation (in which the behavior of the model can be represented using animated graphics). Other informal techniques are the comparison with other models (i.e., the experimental results are compared with the results from other existing models) or the opinions of experts.
- *Static* techniques are used to validate the design of the model and the source code (they do not execute the model). These techniques include the analysis of data flow, analysis based on graphics, and syntactic and semantic analyses.
- *Dynamic* techniques require the execution of the model and validation of its behavior. Some of the techniques include the symbolic execution of the model, cause–effect graphs, regression tests, and stress testing. Other techniques include reachability analysis, degenerative testing (inaccurate or limited input data are used to test the model), and sensitivity analysis (which consists of changing the values of the input as well as those of some

internal parameters in order to observe the reaction of the model to these variations). Varied statistical techniques (including histograms, confidence intervals, or hypothesis tests) are commonly used.

- *Formal* techniques try to prove the correctness of the model using mathematical techniques. They present a high degree of difficulty and serve as a base for other techniques of V&V. This can include induction techniques, inference models, deduction logic, predictive calculus, correction proofs, bisimulation, and model checking.

1.8 SUMMARY

In this chapter, we have given basic ideas and concepts on modeling and simulation. We have presented a brief historical perspective in the field, showing different techniques used for problem solving. We clearly showed the concepts of a system (a formalization of entities under study), the separation between them, and the models we use to reason about the system itself. We briefly introduced different methods to define simulators that will drive the execution of the models (in algorithmic fashion or using specialized devices, including digital computers).

We introduced the concept of the experimental framework, which gives an environment for defining the model, conducting experiments, collecting data, and executing the simulation analysis. The clear separation between the system of interest, the model that represents it, the simulator that executes it, and the experimental frames to conduct tests will make the creation, modification, and use of the simulation tools much easier.

Finally, we introduced varied discrete-event modeling techniques and discussed different classifications, focusing on discrete-event modeling and simulation methodologies (the main focus of this book). We then introduced some term definitions, presented a life cycle proposal for M&S studies, and gave a brief introduction to ideas on verification and validation activities.

REFERENCES

1. Taylor, M. 1996. *Partial differential equations: Basic theory.* New York: Springer–Verlag.
2. Bacon, D. H. 1989. *BASIC heat transfer.* London: Butterworth & Co. Ltd.
3. Lapidus, L., and G. F. Pinder. 1982. *Numerical solution of partial differential equations in science and engineering.* New York: Wiley.
4. Brenan, K. E., S. L. Campbell, and L. R. Petzold. 1989. *Numerical solution of initial-value problems in differential algebraic equations.* New York: Elsevier.
5. Coward, D. Analog computer museum and history center. http://dcoward.best.vwh.net/analog/. Accessed: September 1, 2007.
6. Cassandras, C. G. 1993. *Discrete event systems: Modeling and performance analysis.* Homewood, IL: Aksen: Irwin.
7. Hopcroft, J. E., R. Motwani, and J. D. Ullman. 2007. *Introduction to automata theory, languages, and computation,* 3rd ed. Boston: Pearson/Addison–Wesley.
8. Ho, Y. C. 1989. Introduction to special issue on dynamics of discrete event systems. *Proceedings of the IEEE,* 77:3–6.
9. Birtwistle, G. M. 1979. *A system for discrete event modeling on SIMULA.* London: Macmillan.
10. IFIP Technical Committee 2—Programming. 1968. Simulation programming languages. *Proceedings of the IFIP Working Conference on Simulation Programming Languages,* Oslo, Norway.
11. Pritsker, A. B., and C. D. Pegden. 1979. *Introduction to simulation and SLAM.* New York: Wiley (distributed by Halsted Press).
12. Zeigler, B. P. 1976. *Theory of modeling and simulation.* New York: Wiley-Interscience.
13. Fishwick, P. A. 1995. *Simulation model design and execution: Building digital worlds.* Englewood Cliffs, NJ: Prentice Hall.
14. Zeigler, B. 1984. *Multifaceted modeling and discrete event simulation.* New York: Academic Press.
15. Zeigler, B. P. 1990. *Object-oriented simulation with hierarchical, modular models: Intelligent agents and endomorphic systems.* Boston: Academic Press.

16. Zeigler, B. P. 1998. DEVS theory of quantization. Technical report, DARPA contract N6133997K-0007, ECE Department, the University of Arizona, Tucson.
17. Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation,* 2nd. ed. New York: Academic Press.
18. Zeigler, B. P. 2005. Continuity and change (activity) are fundamentally related in DEVS simulation of continuous systems. *Proceedings of AIS 2004, Artificial Intelligence, Simulation and Planning,* Jeju Island, Korea, 1–17.
19. Kofman, E. 2004. Discrete event simulation of hybrid systems. *SIAM Journal of Scientific and Statistical Computing* 25:1771–1797.
20. Cellier, F. E., and E. Kofman. 2006. *Continuous system simulation.* New York: Springer Science+ Business Media.
21. Alur, R., and D. L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126:183–235.
22. Papadimitriou, C. H. 1994. *Computational complexity.* Reading, MA: Addison–Wesley.
23. Cao, X. R., and Y. C. Ho. 1990. Models of discrete event dynamic systems. *IEEE Control Systems Magazine* 10:69.
24. Glynn, P. W. 1989. A GSMP formalism for discrete event systems. *Proceedings of the IEEE* 77:14–23.
25. Peterson, J. L. 1981. *Petri net theory and the modeling of systems.* Englewood Cliffs, NJ: Prentice Hall.
26. Woodside, C. M. 1988. Throughput calculation for basic stochastic rendezvous networks. *Performance Evaluation* 9:143–160.
27. Rolia, J., and K. C. Sevkik. 1995. The method of layers. *IEEE Transactions on Software Engineering* 21:682–688.
28. Woodside, C. M., S. Majumdar, J. E. Neilson, D. C. Petriu, J. Rolia, A. Hubbard, and G. Franks. 1995. A guide to performance modeling of distributed client–server software systems with layered queuing networks. Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada.
29. Milner, R. 1982. *A calculus of communicating systems.* New York: Springer–Verlag.
30. Manna, Z., and A. Pnueli. 1992. *The temporal logic of reactive and concurrent systems.* New York: Springer–Verlag.
31. Hoare, C. A. R. 1985. *Communicating sequential processes.* Englewood Cliffs, NJ: Prentice Hall International.
32. Harel, D., and M. Politi. 1998. *Modeling reactive systems with statecharts.* New York: McGraw–Hill.
33. Rumbaugh, J., I. Jacobson, and G. Booch. 1999. *The unified modeling language reference manual.* Essex, UK: Addison-Wesley Longman Ltd..
34. Belina, F., and D. Hogrefe. 1989. The CCITT-specification and description language SDL. *Computer Networks ISDN Systems* 16:311–341.
35. Schruben, L. W., T. M. Roeder, W. K. Chan, P. Hyden, and M. Freimer. 2003. Advanced event scheduling methodology. *Proceedings of WSC '03: Proceedings of the 35th Winter Simulation Conference,* New Orleans, 159–165.
36. von Bertalanffy, L. 1969. *General system theory: Foundations, development, applications.* New York: G. Braziller.
37. Nance, R. E. 1981. The time and state relationships in simulation modeling. *Communications of the ACM* 24:173–179.
38. Sargent, R. G. 2005. Verification and validation of simulation models. *Proceedings of WSC '05: Proceedings of the 37th Winter Simulation Conference,* 130–143, Orlando.
39. Law, A. M., and W. D. Kelton. 2000. *Simulation modeling and analysis,* 3rd ed. Boston: McGraw–Hill.
40. Balci, O. 1994. Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Proceedings of WSC '94: Proceedings of the 26th Winter Simulation Conference,* 215–220, Orlando.
41. Lutz, R. 1999. FEDEP V1.4: An update to the HLA process model. *Proceedings of WSC '99: Proceedings of the 31st Winter Simulation Conference,* Phoenix, AZ, 1044–1049.
42. Banks, J., J. S. Carson, B. L. Nelson, and D. Nicol. 2005. *Discrete-event system simulation,* 4th ed. Upper Saddle River, NJ: Prentice Hall.
43. Kleijnen, J. P. C. 2005. Forty years of statistical design and analysis of simulation experiments (DASE). *Proceedings of WSC '05: Proceedings of the 37th Winter Simulation Conference,* Orlando, 1–17.
44. Leemis, L. M., and S. K. Park. 2005. *Discrete-event simulation: A first course.* Upper Saddle River, NJ: Prentice Hall.
45. Pace, D. K. 1993. Naval modeling and simulation verification, validation, and accreditation. *Proceedings of WSC '93: Proceedings of the 25th Winter Simulation Conference,* Los Angeles, 1077–1080.

46. Beizer, B. 1990. *Software testing techniques,* 2nd. ed. New York: Van Nostrand Reinhold Co.
47. Sargent, R. G., P. A. Glasow, J. P. C. Kleijnen, A. M. Law, I. McGregor, and S. Youngblood. 2000. Strategic directions in verification, validation, and accreditation research. *Proceedings of WSC '00: Proceedings of the 32nd Winter Simulation Conference,* Orlando, 909–916.

Modeling and Simulation Concepts

Taylor, M. 1996. Partial differential equations: Basic theory. New York: Springer–Verlag.

Bacon, D. H. 1989. BASIC heat transfer. London: Butterworth & Co. Ltd.

Lapidus, L. , and G. F. Pinder . 1982. Numerical solution of partial differential equations in science and engineering. New York: Wiley.

Brenan, K. E. , S. L. Campbell , and L. R. Petzold . 1989. Numerical solution of initial-value problems in differential algebraic equations. New York: Elsevier.

Coward, D. Analog computer museum and history center. http://dcoward.best.vwh.net/analog/. Accessed: September 1, 2007.

Cassandras, C. G. 1993. Discrete event systems: Modeling and performance analysis. Homewood, IL: Aksen: Irwin.

Hopcroft, J. E. , R. Motwani , and J. D. Ullman . 2007. Introduction to automata theory, languages, and computation, 3rd ed. Boston: Pearson/Addison–Wesley.

Ho, Y. C. 1989. Introduction to special issue on dynamics of discrete event systems. Proceedings of the IEEE, 77:3–6.

Birtwistle, G. M. 1979. A system for discrete event modeling on SIMULA. London: Macmillan.

IFIP Technical Committee 2—Programming . 1968. Simulation programming languages. Proceedings of the IFIP Working Conference on Simulation Programming Languages, Oslo, Norway.

Pritsker, A. B. , and C. D. Pegden . 1979. Introduction to simulation and SLAM. New York: Wiley (distributed by Halsted Press).

Zeigler, B. P. 1976. Theory of modeling and simulation. New York: Wiley-Interscience.

Fishwick, P. A. 1995. Simulation model design and execution: Building digital worlds. Englewood Cliffs, NJ: Prentice Hall.

Zeigler, B. 1984. Multifaceted modeling and discrete event simulation. New York: Academic Press.

Zeigler, B. P. 1990. Object-oriented simulation with hierarchical, modular models: Intelligent agents and endomorphic systems. Boston: Academic Press.

Zeigler, B. P. 1998. DEVS theory of quantization. Technical report, DARPA contract N6133997K-0007, ECE Department, the University of Arizona, Tucson.

Zeigler, B. P. , H. Praehofer , and T. G. Kim . 2000. Theory of modeling and simulation, 2nd. ed. New York: Academic Press.

Zeigler, B. P. 2005. Continuity and change (activity) are fundamentally related in DEVS simulation of continuous systems. Proceedings of AIS 2004, Artificial Intelligence, Simulation and Planning, Jeju Island, Korea, 1–17.

Kofman, E. 2004. Discrete event simulation of hybrid systems. SIAM Journal of Scientific and Statistical Computing 25:1771–1797.

Cellier, F. E. , and E. Kofman . 2006. Continuous system simulation. New York: Springer Science+ Business Media.

Alur, R. , and D. L. Dill . 1994. A theory of timed automata. Theoretical Computer Science 126:183–235.

Papadimitriou, C. H. 1994. Computational complexity. Reading, MA: Addison–Wesley.

Cao, X. R. , and Y. C. Ho . 1990. Models of discrete event dynamic systems. IEEE Control Systems Magazine 10:69.

Glynn, P. W. 1989. A GSMP formalism for discrete event systems. Proceedings of the IEEE 77:14–23.

Peterson, J. L. 1981. Petri net theory and the modeling of systems. Englewood Cliffs, NJ: Prentice Hall.

Woodside, C. M. 1988. Throughput calculation for basic stochastic rendezvous networks. Performance Evaluation 9:143–160.

Rolia, J. , and K. C. Sevkik . 1995. The method of layers. IEEE Transactions on Software Engineering 21:682–688.

Woodside, C. M. , S. Majumdar , J. E. Neilson , D. C. Petriu , J. Rolia , A. Hubbard , and G. Franks . 1995. A guide to performance modeling of distributed client–server software systems with layered queuing networks. Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada.

Milner, R. 1982. A calculus of communicating systems. New York: Springer–Verlag.

Manna, Z. , and A. Pnueli . 1992. The temporal logic of reactive and concurrent systems. New York: Springer–Verlag.

Hoare, C. A. R. 1985. Communicating sequential processes. Englewood Cliffs, NJ: Prentice Hall International.

Harel, D. , and M. Politi . 1998. Modeling reactive systems with statecharts. New York: McGraw–Hill.

Rumbaugh, J. , I. Jacobson , and G. Booch . 1999. The unified modeling language reference manual. Essex, UK: Addison-Wesley Longman Ltd..

Belina, F. , and D. Hogrefe . 1989. The CCITT-specification and description language SDL. Computer Networks ISDN Systems 16:311–341.

Schruben, L. W. , T. M. Roeder , W. K. Chan , P. Hyden , and M. Freimer . 2003. Advanced event scheduling methodology. Proceedings of WSC ’03: Proceedings of the 35th Winter Simulation Conference, New Orleans, 159–165.

von Bertalanffy, L. 1969. General system theory: Foundations, development, applications. New York: G. Braziller.

Nance, R. E. 1981. The time and state relationships in simulation modeling. Communications of the ACM 24:173–179.

Sargent, R. G. 2005. Verification and validation of simulation models. Proceedings of WSC ’05: Proceedings of the 37th Winter Simulation Conference, 130–143, Orlando.

Law, A. M. , and W. D. Kelton . 2000. Simulation modeling and analysis, 3rd ed. Boston: McGraw–Hill.

Balci, O. 1994. Validation, verification, and testing techniques throughout the life cycle of a simulation study. Proceedings of WSC ’94: Proceedings of the 26th Winter Simulation Conference, 215–220, Orlando.

Lutz, R. 1999. FEDEP V1.4: An update to the HLA process model. Proceedings of WSC ’99: Proceedings of the 31st Winter Simulation Conference, Phoenix, AZ, 1044–1049.

Banks, J. , J. S. Carson , B. L. Nelson , and D. Nicol . 2005. Discrete-event system simulation, 4th ed. Upper Saddle River, NJ: Prentice Hall.

Kleijnen, J. P. C. 2005. Forty years of statistical design and analysis of simulation experiments (DASE). Proceedings of WSC ’05: Proceedings of the 37th Winter Simulation Conference, Orlando, 1–17.

Leemis, L. M. , and S. K. Park . 2005. Discrete-event simulation: A first course. Upper Saddle River, NJ: Prentice Hall.
Pace, D. K. 1993. Naval modeling and simulation verification, validation, and accreditation. Proceedings of WSC '93: Proceedings of the 25th Winter Simulation Conference, Los Angeles, 1077–1080.
Beizer, B. 1990. Software testing techniques, 2nd. ed. New York: Van Nostrand Reinhold Co.
Sargent, R. G. , P. A. Glasow , J. P. C. Kleijnen , A. M. Law , I. McGregor , and S. Youngblood . 2000. Strategic directions in verification, validation, and accreditation research. Proceedings of WSC '00: Proceedings of the 32nd Winter Simulation Conference, Orlando, 909–916.

Introduction to the DEVS Modeling and Simulation Formalism

Zeigler, B. P. 1976. Theory of modeling and simulation. New York: Wiley-Interscience.
Klir, G. J. 1972. Trends in general systems theory. New York: Wiley-Interscience.
Zadeh, L. A. , and C. A. Desoer . 1963. Linear system theory: The state space approach. New York: McGraw–Hill.
Zeigler, B. P. , H. Praehofer , and T. G. Kim . 2000. Theory of modeling and simulation, 2nd. ed. New York: Academic Press.
Zeigler, B. 1984. Multifaceted modeling and discrete event simulation. New York: Academic Press.
Cellier, F. E. , and E. Kofman . 2006. Continuous system simulation. New York: Springer Science+ Business Media.
Nutaro, J. 2003. Parallel discrete event simulation with application to continuous systems. PhD thesis, University of Arizona, Tucson.
Nutaro, J. , and H. Sarjoughian . 2004. Design of distributed simulation environments: A unified system-theoretic and logical processes approach. Simulation 80:577–589.
Kim, T. G. , S. M. Cho , and W. B. Lee . 2000. DEVS framework for systems development: Unified specification for logical analysis, performance evaluation and implementation. In Discrete event modeling & simulation: Enabling future technologies, ed. H. S. Sarjoughian and F. Cellier . New York: Springer–Verlag.
Chow, A. C. , and B. Zeigler . 1994. Parallel DEVS: A parallel, hierarchical, modular modeling formalism. Proceedings of Winter Simulation Conference, Orlando, FL.
Barros, F. J. 1997. Modeling formalisms for dynamic structure systems. ACM Transactions on Modeling and Computer Simulation 7:501–515.
Barros, F. 1998. Abstract simulators for the DSDE formalism. Proceedings of Winter Simulation Conference, Washington, D.C., 407–412.
Barros, F. J. 1995. Dynamic structure discrete event system specifications: A new formalism for dynamic structure modeling and simulation. Proceedings of Winter Simulation Conference, Arlington, VA, 781–785.
Uhrmacher, A. M. 2001. Dynamic structure in modeling and simulation: A reflective approach. ACM Transactions on Modeling and Computer Simulation 11:206–232.
Uhrmacher, A. M. , and J. Himmeelspach . 2004. Processing dynamic PDEVS models. Proceedings of 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), Volenlam, the Netherlands.
Pantelides, C. C. 1988. The consistent initialization of differential-algebraic systems. SIAM Journal of Scientific and Statistical Computing 9:213–231.
Fábián, G. D. , D. A. van Beek , and J. E. Rooda . 2000. Substitute equations for index reduction and discontinuity handling. Proceedings of Third IMACS Symposium on Mathematical Modelling, Vienna, Austria.
Press, W. H. , B. P. Flannery , S. A. Teukolsky , and W. T. Vetterling . 1986. Numerical recipes. Cambridge: Cambridge University Press.
Zeigler, B. P. , DEVS theory of quantization. 1998. Technical report, DARPA contract N6133997K-0007, ECE Dept., University of Arizona, Tucson.
Kofman, E. 2003. Quantized-state control. A method for discrete event control of continuous systems. Latin American Applied Research Journal 33:339–406.
Giambiasi, N. , B. Escude , and S. Ghosh . 2000. GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. Transactions of the SCS 17:120–134.

The Cell-DEVS Formalism

Chandrupatla, T. , and A. Belegundu . 1997. Introduction to finite elements in engineering. Upper Saddle River, NJ: Prentice Hall.
Wolfram, S. 1986. Theory and applications of cellular automata, vol. 1. Singapore: World Scientific.
Wolfram, S. 2002. A new kind of science. Champaign, IL: Wolfram Media.
Gutowitz, H. 1995. Cellular automata and the sciences of complexity. Parts I–II. Complexity 1:16–22.
Toffoli, T. , and N. Margolus . 1987. Cellular automata machines: A new environment for modeling. Cambridge, MA: MIT Press.
von Neumann, J. 1966. Theory of self-reproducing cellular automata. Urbana: University of Illinois Press.
Wainer, G. , and N. Giambiasi . 2002. N-dimensional cell-DEVS. Discrete Events Systems: Theory and Applications 12:135–157.

Wainer, G. 1998. Discrete-event cellular models with explicit delays. PhD thesis, Université d'Aix-Marseille III, France.
Zeigler, B. P. 1976. Theory of modeling and simulation. New York: Wiley-Interscience.

Introduction to the CD++ Toolkit

Nutaro, J. ADEVS. URL: http://www.ornl.gov/~1qn/adevs/index.html. Accessed: June 1, 2007.
Kim, K. H. , Y. R. Seong , T. G. Kim , and K. H. Park . 1996. Distributed simulation of hierarchical DEVS models: Hierarchical scheduling locally and time warp globally. Transactions of the SCS 13 (3): 135–154.
Zeigler, B. , Y. Moon , D. Kim , and D. Kim . 1996. DEVS-C++: A high performance modeling and simulation environment. Proceedings of 29th Hawaii International Conference on System Sciences, Honolulu.
Zeigler, B. P. 1990. Object-oriented simulation with hierarchical, modular models: Intelligent agents and endomorphic systems. Boston: Academic Press.
Zeigler, B. , and D. Kim . 1995. Extending the DEVS-scheme knowledge-based simulation environment for real-time event-based control. Technical report, Department of Electrical and Computer Engineering, University of Arizona.
Sarjoughian, H. S. , and B. P. Zeigler . 2000. DEVS and HLA: Complementary paradigms for M&S? Transactions of the SCS 17:187–197.
Zeigler, B. P. 1999. Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions. Simulation Interoperability Workshop, Orlando, FL.
IEEE Std 1516.1-2000 . 2001. IEEE standard for modeling and simulation. High level architecture (HLA)—Federate interface specification. IEEE Std 1516.1-2000: i–467.
Sarjoughian, H. S. , and B. P. Zeigler . 1998. DEVSJAVA: Basis for a DEVS-based collaborative M&S environment. Proceedings of SCS International Conference on Web-Based Modeling and Simulation, San Diego, CA.
Kim, T. G. 1994. DEVSIM++ user's manual. CORE Lab, EE Dept, KAIST, Taejon, Korea.
Dávila, J. , and M. Uzcágegui . 2000. GALATEA: A multi-agent, simulation platform. Proceedings of International Conference on Modeling, Simulation and Neural Networks, Mérida, Venezuela.
Himmelspach, J. , and A. Uhrmacher . 2004. A component-based simulation layer for JAMES. Proceedings of 18th Workshop on Parallel and Distributed Simulation (PADS), Kufstein, Austria, 115–122.
Filippi, J. B. , and P. Bisgambiglia . 2004. JDEVS: An implementation of a DEVS based formal framework. Environmental Modeling and Software 19:261–274.
Bolduc, J. S. , and H. Vangheluwe . 2001. The modeling and simulation package PythonDEVS for classical hierarchical DEVS. Technical report MSDL-TR-2001-01, McGill University.
de Lara, J. , and H. Vangheluwe . 2002. AToM3: A tool for multi-formalism and meta-modeling. Proceedings of Fundamental Approaches to Software Engineering, 5th International; Lecture Notes in Computer Science, 174–188.
Praehofer, H. , and G. Reisinger . 1995. Object-oriented realization of a parallel discrete event simulator. Technical report, Johannes Kepler University, Department of System Theory and Information Engineering.
Henning, M. , and S. Vinoski . 1999. Advanced CORBA programming with C++. Reading, MA: Addison–Wesley.
Dongarra, J. J. 1995. High performance computing: Technology, methods and applications. Amsterdam: Elsevier.
Wainer, G. 2002. CD++: A toolkit to develop DEVS models. Software Practice and Experience 32:261.
Plauger, P. J. , A. Stepanov , M. Lee , and D. Musser . 2000. The C++ standard template library. Upper Saddle River, NJ: Prentice Hall.
Gardner, M. 1970. The fantastic combinations of John Conway's new solitaire game "Life." Scientific American 23:120–123.
Christen, G. , A. Dobniewski , and G. Wainer . 2004. Modeling state-based DEVS models CD++. Proceedings of MGA, Advanced Simulation Technologies Conference 2004, Arlington, VA.
Christen, G. , A. Dobniewski , and G. Wainer . 2001. Defining DEVS models with the CD++ toolkit. Proceedings of European Simulation Symposium, Marseilles, France.
Praehofer, H. , and D. Pree . 1993. Visual modeling of DEVS-based multiformalism systems based on higraphs. Proceedings of WSC '93, the 25th Winter Simulation Conference, Los Angeles, CA, 595–603.
Kidisyuk, K. , and G. Wainer . 2007. CD++Modeler: A graphical viewer for DEVS models. Technical report SCE-017, Ottawa, ON, Canada.

Modeling Simple DEVS and Cell-DEVS Models in CD++

Nayfeh, B. 1993. Cellular automata for solving mazes. Doctor Dobb's Journal, February 1993.
Lam, K. , and G. Wainer . 2003. Modeling of maze-solving problems using cell-DEVS. Proceedings of 2003 SCS Summer Computer Simulation Conference, Montreal, QC, Canada.
Toffoli, T. 1994. Occam, Turing, von Neumann, Jaynes: How much can you get for how little? (A conceptual introduction to cellular automata). Proceedings of International Conference on Cellular Automata for Research and Industry, Rende, Italy.
Toffoli, T. , and N. Margolus . 1987. Cellular automata machines: A new environment for modeling. Cambridge, MA: MIT Press.

Liu, Q. , and G. Wainer . 2005. Simulating market dynamics with CD++. Proceedings of International Conference on Computational Science, Atlanta, GA, 368–372.
Oda, S. H. , K. Iyori , M. Ken , and K. Ueda . 1999. The application of cellular automata to the consumer's problem. Proceedings of Second Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98, Canberra, Australia, 454–461.
Ameghino, J. , and G. Wainer . 2004. Application of the cell-DEVS formalism for modeling cell spaces. Proceedings of Artificial Intelligence, Simulation and Planning, Jeju Island, Korea.
Wainer, G. , and N. Giambiasi . 2001. Timed cell-DEVS: Modelling and simulation of cell spaces. In Discrete event modeling & simulation: Enabling future technologies, ed. H. S. Sarjoughian and F. E. Cellier . New York: Springer–Verlag.

Discrete-Event Applications with DEVS

Kidisyuk, K. , and G. Wainer . 2007. CD++Builder: A toolkit to develop DEVS models. Proceedings of DEVS Symposium 2007, Norfolk, VA.
Cidre, J. I. 2006. A Web-based interface for the CD++Modeler toolkit. MCS thesis, Universidad de Buenos Aires, Argentina.
Kidisyuk, K. , and G. Wainer . 2007. CD++Modeler: A graphical viewer for DEVS models. Technical report SCE-017, Ottawa, Carleton University, Canada, 2007 (in Poster Sessions, SpringSim 2008, Ottawa, Canada).
Christen, G. , A. Dobniewski , and G. Wainer . 2001. Defining DEVS models with the CD++ toolkit. Proceedings of European Simulation Symposium, Marseilles, France.
Christen, G. , A. Dobniewski , and G. Wainer . 2004. Modeling state-based DEVS models CD++. Proceedings of MGA, Advanced Simulation Technologies Conference 2004, Arlington, VA.
Christen, G. , and A. Dobniewski . 2003. Extending the CD++ toolkit to define DEVS graphs. MSc thesis, Computer Science Dept., Universidad de Buenos Aires, Argentina.

Defining Varied Modeling Techniques Using DEVS

Vangheluwe, H. L. M. 2000. DEVS as a common denominator for multiformalism hybrid systems modeling. Proceedings of Computer-Aided Control System Design, 2000, IEEE International Symposium on CACSD 2000, Anchorage, AK, 129–134.
Zeigler, B. P. 1976. Theory of modeling and simulation. New York: Wiley-Interscience.
Zeigler, B. P. , and S. Vahie . 1993. DEVS formalism and methodology: Unity of conception/diversity of application. Proceedings of WSC '93: Proceedings of the 25th Winter Simulation Conference, Los Angeles, CA, 573–579.
Zheng, T. , and G. Wainer . 2003. Implementing finite state machines using the CD++ toolkit. Proceedings of the 2003 SCS Summer Computer Simulation Conference, Montreal, Quebec, Canada.
Peterson, J. L. 1981. Petri net theory and the modeling of systems. Englewood Cliffs, NJ: Prentice Hall.
Jacques, C. , and G. Wainer . 2002. Using the CD++ DEVS toolkit to develop Petri nets. Proceedings of Summer Computer Simulation Conference, San Diego, CA.
Neilson, J. E. , C. M. Woodside , D. C. Petriu , and S. Majumdar . 1995. Software bottlenecking in client–server systems and rendez-vous networks. IEEE Transactions on Software Engineering 21 (9): 776–782.
Woodside, C. M. , J. E. Neilson , D. C. Petriu , and S. Majumdar . 1995. The stochastic rendezvous network model for performance of synchronous client–server-like distributed software. IEEE Transactions on Computers 44 (1): 20–34.
Woodside, C. M. , S. Majumdar , J. E. Neilson , D. C. Petriu , J. Rolia , A. Hubbard , and G. Franks . 1995. A guide to performance modeling of distributed client-server software systems with layered queuing networks. Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada.
Petriu, D. , and G. Wainer . 2004. A library of layered queuing networks using the DEVS formalism. Proceedings of Mediterranean Multiconference on Modeling and Simulation, Bergeggi, Italy.
Christen, E. , K. Bakalar , A. Dewey , and E. Moser . 1999. DAC'99 VHDL-AMS tutorial. Proceedings of 36th Design Automation Conference, New Orleans, LA.
Mehta, S. , and G. Wainer . 2005. sAMS-VHDL: A tool for modeling hybrid hardware description languages. Proceedings of the 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference, San Diego, CA.
Press, W. H. , B. P. Flannery , S. A. Teukolsky , and W. T. Vetterling . 1986. Numerical recipes. Cambridge: Cambridge University Press.
Taylor, M. 1996. Partial differential equations: Basic theory. New York: Springer–Verlag.
Brenan, K. E. , S. L. Campbell , and L. R. Petzold . 1989. Numerical solution of initial-value problems in differential algebraic equations. New York: Elsevier.
Åström, K. J. , H. Elmqvist , and S. E. Mattsson . 1998. Evolution of continuous-time modeling and simulation. 12th European Simulation Multiconference, ESM'98, Manchester, UK.
D'Abreu, M. , and G. Wainer . 2003. Defining hybrid system models using DEVS quantization techniques. Proceedings of the Winter Simulation Conference, New Orleans, LA.
Cellier, F. E. , and E. Kofman . 2006. Continuous system simulation. New York: Springer Science+ Business Media.

Samantaray, A. 2007. About bond graph—The system modeling world. URL: http://www.bondgraph.info/about.html
Paul, C. R. 2001. Fundamentals of electric circuit analysis. New York: John Wiley & Sons.
Kofman, E. 2003. Discrete event based simulation and control of continuous systems. PhD thesis, Universidad Nacional de Rosario, Argentina.
Banerjee, S. 2003. Dynamics of physical systems—The language of bond graphs. URL: http://www.Ee.Iitkgp.Ernet.In/~soumitro/dops/chap4.Pdf
Banerjee, S. 2005. Dynamics for Engineers. New York: Wiley.
D'Abreu, M. , and G. Wainer . 2005. M/CD++: Modeling continuous systems using Modelica and DEVS. Proceedings of MASCOTS 2005, Atlanta, GA.
D'Abreu, M. 2004. Defining a compiler for discrete-event simulation of continuous systems. MSc thesis, Computer Science Dept., Universidad de Buenos Aires, Argentina.
Karnopp, D. , D. Margolis , and R. Rosenber . 1990. System dynamics: A unified approach. New York: Wiley-Interscience.
Dynasim Laboratories . 2004. Dymola. Available online via: http://www.Dynasim.com/dymola.htm
Petzold, L. R. 1993. A description of DASSL: A differential/algebraic system solver. IMACS Transactions Scientific Computing 1:65–68.
Mehta, S. , and G. Wainer . 2005. SAMS-VHDL: A tool for modeling hybrid hardware description languages. Proceedings of the 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference. San Diego, CA.

Applications in Biology

Benfenati, F. , F. Valtorta , and P. Greengard . 1991. Computer modeling of synapsin I binding to synaptic vesicles and F-actin: Implications for regulation of neurotransmitter release. Proceedings of the National Academy of Sciences USA 88:575–579.
Bain, R. , S. Jafer , M. Dumontier , G. Wainer , and J. Cheetham . 2006. Vesicle, synapsin and actin concentration time series modelling at the presynaptic nerve terminal (poster). Proceedings of Symposium on Progress in Systems Biology 2006, Ottawa, ON, Canada.
Wainer, G. , B. Al-aubidy , A. Dias , R. Bain , S. Jafer , M. Dumontier , and J. Cheetham . 2007. Advanced DEVS models with applications to biomedicine. Proceedings of AIS'2007 Artificial Intelligence, Simulation and Planning, Buenos Aires, Argentina.
Hunt, C. A. , G. Ropella , M. Roberts , and L. Yan . 2005. Biomimetic in silico devices. Proceedings of Computational Methods in Systems Biology 2004; Lecture Notes in Bioinformatics 3082, 35–43.
Ameghino, J. , E. Glinsky , and G. Wainer . 2003. Applying cell-DEVS models of complex systems. Proceedings of Summer Computer Simulation Conference, Montreal, QC, Canada.
Shang, H. , and G. Wainer . 2005. A model of virus spreading in CD++. Proceedings of the International Conference on Computational Science, Atlanta, GA.
Dzwinel, W. 2004. A cellular automata model of population infected by periodic plague. Proceedings of ACRI 2004, LNCS 3305, 464–473.
Giambiasi, N. , and G. Wainer . 2005. Using G-DEVS and cell-DEVS to model complex continuous systems. Simulation: Transactions of the Society for Modeling and Simulation International 81:137–151.
Goldschlager, N. , and M. Goldman . 1989. Principles of clinical electrocardiography. Norwalk, CT: Appleton and Lange.
Alberts, B. , D. Bray , L. Lewis , M. Raff , K. Roberts , and D. Watson . 1983. Molecular biology of the cell, 1st ed. New York: Garland Publishing, Inc.
Hodgkin, A. , and A. Huxley . 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. Journal of Physiology 117:500–544.
Saxberg, B. , and R. Cohen . 1991. Cellular automata models of cardiac conduction. In Theory of heart, edited by L. Glass , P. Hunter , and A. McCulloch . New York: Springer–Verlag.
Fenton, F. 2000. Numerical simulations of cardiac dynamics. What can we learn from simple and complex models? IEEE Computers in Cardiology 27:251–254.
Wainer, G. 2004. Performance analysis of continuous cell-DEVS models. Proceedings of High Performance Computing & Simulation (HPC&S) Conference, 18th European Simulation Multiconference, Magdeburg, Germany.
Press, W. H. , B. P. Flannery , S. A. Teukolsky , and W. T. Vetterling . 1986. Numerical recipes. Cambridge: Cambridge University Press.
Djafarzadeh, R. , T. Mussivand , and G. Wainer . 2005. Modeling energy pathways in cells. Proceedings of 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference, San Diego, CA.
Krauss, S. 2001. Mitochondria: Structure and role in respiration. In Nature encyclopedia of life sciences. New York: Nature Publishing Group.
Poulton, J. , and L. Bindoff . 2000. Mitochondrial respiratory chain disorders. In Nature encyclopedia of life sciences. New York: Nature Publishing Group.
Curtis, H. , and N. Barnes . 1989. Biology, 5th ed. New York: W. H. Freeman.
Khan, A. , G. Wainer , W. Venhola , and M. Jemtrud . 2005. On the use of CD++/Maya for visualization of discrete-event models. Proceedings of IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, Paris.

Models in Defense and Emergency Planning

Palmore, J. 1997. Mini-symposium/workshop report. Warfare analysis and complexity. Military Operations Research Society. September 15–17, 1997. http://www.mors.org, JHU/APL. Laurel, MD.

Madhoun, R. , and G. Wainer . 2005. Developing defense applications using DEVS/cell-DEVS. Journal of Defense Modeling and Simulation 2:121–143.

MacSween, P. , and G. Wainer . 2005. On the construction of complex models using reusable components. 2004 Spring Simulation Interoperability Workshop, Arlington, VA.

Reynolds, C. W. 1999. Steering behaviors for autonomous characters. Proceedings of Game Developers Conference, San Jose, CA.

Woodcock, A. E. R. , L. Cobb , and J. Dockery . 1988. CA: A new method for battlefield simulation. Signal 42:41–50.

Gore, J. 1996. Chaos, complexity and the military. Technical report 96-E-61, National Defense University, National War College. Military Strategy and Operation Seminar D.

Ilachinski, A. 2004. Artificial war: Multiagent-based simulation of combat. Singapore: World Scientific Press.

Ilachinski, A. 2000. Irreducible semi-autonomous adaptive combat (ISAAC)—An artificial life approach to land combat. Military Operation Research 5:29–46.

López, A. , and G. Wainer . 2004. Improved cell-DEVS model definition in CD++. In ACRI 2004, LNCS 3305, ed. P. M. A. Sloot , B. Chopard , and A. G. Hoekstra . New York: Springer–Verlag.

Kim, H. , D. Lee , J. H. Park , J. G. Lee , B. J. Park , and S. H. Lee . 2001. Establishing the methodologies for human evacuation simulation in marine accidents. Proceedings of 29th Conference on Computers and Industrial Engineering, Montréal, QC, Canada.

Ameghino, J. , and G. Wainer . 2004. Application of the cell-DEVS formalism for modeling cell spaces. In Proceedings of Artificial Intelligence, Simulation and Planning, Jeju Island, Korea, LNCS 3397.

Ameghino, J. , E. Glinsky , and G. Wainer . 2003. Applying cell-DEVS models of complex systems. In Proceedings of Summer Computer Simulation Conference, Montreal, QC, Canada.

Brunstein, M. , and J. Ameghino . 2003. Modeling evacuation processes using Cell-DEVS. Internal Report. Universidad de Buenos Aires, Computer Science Department.

Models in Architecture and Construction

Halpin, D. W. , and L. S. Riggs . 1992. Planning and analysis of construction operations. New York: Wiley Interscience.

Kamat, V. R. 2001. Visualizing simulated construction operations in 3D. Journal of Computing in Civil Engineering 15(4):329–337.

Hajjar, D. , and S. AbouRizk . 1999. Simphony: An environment for building special purpose construction simulation tools. Proceedings of 1999 Winter Simulation Conference, Phoenix, AZ.

Malamud, B. , and D. Turcotte . 2000. Cellular-automata models applied to natural hazards. Earth System Science May/June: 42–51.

Christensen, K. 1991. Dynamical and spatial aspects of sandpile cellular automata. Journal of Statistical Physics 63:653.

Pla-Castells, M. , I. García , and R. J. Martínez . 2004. Granular system models for real-time simulation. Industrial Simulation Conference, Malaga, Spain, 88–93.

Saadawi, H. , and G. Wainer . 2003. Modeling a sand pile application using cell-DEVS. Proceedings of 2003 SCS Summer Computer Simulation Conference, Montreal, QC, Canada.

Zhang, C. , T. M. Zayed , A. Hammad , and G. Wainer . 2005. Representation and analysis of spatial resources in construction simulation. Proceedings of WSC ’05: Proceedings of the 37th Conference on Winter Simulation, Orlando, FL, 1541–1548.

Klüpfel, H. , T. Meyer-König , J. Wahle , and M. Schreckenberg . 2000. Microscopic simulation of evacuation processes on passenger ships. Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry, Karlsruhe, Germany, 63–71.

Kim, H. , D. Lee , J. H. Park , J. G. Lee , B. J. Park , and S. H. Lee . 2001. Establishing the methodologies for human evacuation simulation in marine accidents. Proceedings of 29th Conference on Computers and Industrial Engineering, Montreal, QC, Canada.

Ameghino, J. , and G. Wainer . 2004. Application of the cell-DEVS formalism for modeling cell spaces. Proceedings of Artificial Intelligence, Simulation and Planning, Jeju Island, Korea, LNCS 3397.

Poliakov, E. , G. Wainer , and M. Jemtrud . 2007. A busy day at the SAT building. Proceedings of AIS 2007, Artificial Intelligence, Simulation and Planning, Buenos Aires, Argentina.

Meyer-König, T. , H. Klüpfel , and M. Schreckenberg . 2001. A microscopic model for simulating mustering and evacuation processes onboard passenger ships. Proceedings of TIEMS, the International Emergency Management Society’s Eighth Annual Conference, Oslo, Norway.

Models in Environmental Sciences

Darwen, P. J. , and D. G. Green . 1996. Viability of populations in a landscape. Ecological Modeling 85:165.

Wainer, G. 2006. Applying cell-DEVS methodology for modeling the environment. Simulation: Transactions of the Society for Modeling and Simulation International 82:635–660.

Langton, C. 1986. Studying artificial life with cellular automata. Physica 22D:120–149.

Ameghino, J. , and G. Wainer . 2000. Application of the cell-DEVS paradigm using N-CD++. Proceedings of the 32nd SCS Summer Computer Simulation Conference, Vancouver, Canada.

Nishidate, K. , M. Baba , and R. Gaylord . 1996. Cellular automaton model for random walkers. Physical Review Letters 77:1675–1678.

Ameghino, J. , E. Glinsky , and G. Wainer . 2003. Applying cell-DEVS models of complex systems. Proceedings of 35th Summer Computer Simulation Conference, Montreal, QC, Canada.

DEVS Representation of Spatially Distributed Systems: Validity, Complexity Reduction . 1996. Proceedings 6th AI, Simulation and Planning in High Autonomy Systems, San Diego, CA.

Ameghino, J. , A. Troccoli , and G. Wainer . 2001. Modeling and simulation of complex physical systems using cell-DEVS. Proceedings of 34th IEEE/SCS Annual Simulation Symposium, Seattle, WA.

Bianchini, A. , F. Indovina , and E. Rinaldi . 1999. Cellular automata for the study of the diffusion of pollutants within the basins of the lagoon: The case of the Venetian lagoon. Proceedings of 6th International Conference on Computers in Urban Planning and Urban Management, Venice, Italy.

Bandini, S. , and G. Pavesi . 2002. Simulation of vegetable population dynamics based on cellular automata. Proceedings of 5th International Conference on Cellular Automata for Research and Industry. Geneva, Switzerland, LNCS 2493.

Broadbent, S. R. , and J. M. Hammersley . 1957. Percolation processes. I. Crystals and mazes. Proceedings of the Cambridge Philosophical Society 53:629–641.

Rothermel, R. 1972. A mathematical model for predicting fire spread in wildland fuels. Research paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 40 pp.

Vasconcelos, M. 1988. Simulation of fire behavior with a geographical information system. MSc thesis, University of Arizona.

Vasconcelos, M. , J. Pereira , and B. Zeigler . 1995. Simulation of fire growth using discrete event hierarchical modular models. EARSeL. Advances in Remote Sensing 3:54–62.

Ntaimo, L. , B. Khargharia , B. Zeigler , and M. Vasconcelos . 2004. Forest fire spread and suppression in DEVS. Simulation: Transactions of the Society for Modeling and Simulation International 80:479–500.

Muzy, A. , E. Innocenti , A. Aiello , J. Santucci , and G. Wainer . 2005. Discrete-event modeling and simulation of fire spreading across a fuel bed. Simulation: Transactions of the Society for Modeling and Simulation International 81:103–117.

Muzy, A. , T. Marcelli , A. Aiello , P. A. Santoni , J. F. Santucci , and J. H. Balbi . 2001. An object-oriented environment applied to a semiphysical model of fire spread across a fuel bed. Proceedings of European Simulation Symposium 2001—DEVS Workshop, Marseille, France.

Balbi, J. H. , P. A. Santoni , and J. L. Dupuy . 1999. Dynamic modeling of fire spread across a fuel bed. International Journal of Wildland Fire 9:275–284.

López, A. , and G. Wainer . 2004. Improved cell-DEVS model definition in CD++. In ACRI 2004, LNCS 3305, ed. P. M. A. Sloot , B. Chopard , and A. G. Hoekstra . New York: Springer–Verlag.

MacLeod, M. , R. Chreyh , and G. Wainer . 2006. Improved cell-DEVS models for fire spreading analysis. Proceedings of ACRI 2006, LNCS Vol. 4173, Perpignan, France.

Models in Physics and Chemistry

Wolfram, S. 2002. A new kind of science. Champaign, IL: Wolfram Media.

Wolfram, S. 1986. Theory and applications of cellular automata, vol. 1. Singapore: World Scientific.

Toffoli, T. , and N. Margolus . 1987. Cellular automata machines: A new environment for modeling. Cambridge, MA: MIT Press.

Ding, W. , X. Wu , L. Checiu , C. Lin , and G. Wainer . 2005. Definition of cell-DEVS models for complex diffusion systems. Proceedings of Summer Computer Simulation Conference, Philadelphia, PA.

Halsey, T. C. 2001. Diffusion-limited aggregation: A model for pattern formation. Physics Today 54.

Weimar, J. 2002. Three-dimensional cellular automata for reaction diffusion systems. Fundamenta Informaticae 52:275–282.

Checiu, L. , and G. Wainer . 2005. Experimental results on the use of M/CD++. Proceedings of Summer Computer Simulation Conference, Philadelphia, PA.

Gaylord, R. J. , and K. Nishidate . 1996. Modeling nature. New York: Springer–Verlag.

Reiter, C. 2005. A local cellular model for snow crystal growth. Chaos, Solitons & Fractals 23:1111–1119.

Kremeyer, K. 1997. Experimental and computational investigations of binary solidification. PhD thesis, Department of Physics, University of Arizona, Tucson, AZ.

Ameghino, J. , and G. Wainer . 2004. Application of the cell-DEVS formalism for modeling cell spaces. Proceedings of Artificial Intelligence, Simulation and Planning 2004, LNCS 3397, Jeju Island, Korea.
Nutaro, J. 2006. A discrete event method for wave simulation. ACM Transactions Model Computer Simulation 16:174–195.
Andrade, F. J. , F. A. Iñón , M. B. Tudino , and O. E. Troccoli . 1999. Integrated conductimetric detection: Mass distribution in a dynamic sample zone inside a flow injection manifold. Analytica Chimica Acta 379:99–106.
Troccoli, A. , J. Ameghino , F. Iñón , and G. Wainer . 2002. A flow injection model using cell-DEVS. Proceedings of 35th IEEE/SCS Annual Simulation Symposium, San Diego, CA.
Wainer, G. , and B. P. Zeigler . 2000. Experimental results of timed cell-DEVS quantization, AI and simulation. AIS 2000, Tucson, AZ, 203–208.
Zeigler, B. P. 1998. DEVS theory of quantization. Technical report, DARPA Contract N6133997K-0007, ECE Dept., the University of Arizona, Tucson, AZ.
Chandrupatla, T. , and A. Belegundu . 1997. Introduction to finite elements in engineering. Upper Saddle River, NJ: Prentice Hall.
Brauer, J. 1988. What every engineer should know about finite element analysis. New York: Marcel Dekker, Inc.
Saadawi, H. , and G. Wainer . 2003. Improving the finite element method using cell-DEVS. Proceedings of 2003 SCS Summer Computer Simulation Conference, Montreal, QC, Canada.
Saadawi, H. , and G. Wainer . 2007. Defining models of complex 2D physical systems using cell-DEVS. Simulation Modeling Practice and Theory 15:1268–1291.
Saadawi, H. , and G. Wainer . 2004. Modeling complex physical systems using 2D finite element cell-DEVS. Proceedings of MGA, Advanced Simulation Technologies Conference 2004 (ASTC'04), Arlington, VA.
Hardy, J. , O. de Pazzis , and Y. Pomeau . 1976. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. Physical Review A 13:1949–1961.
Rothman, D. A. , and S. Zaleski . 2004. Lattice-gas cellular automata: Simple models of complex hydrodynamics (Collection Alea-Saclay: Monographs and texts in statistical physics). Cambridge: Cambridge University Press.
Kameyama, K. 1997. Virtual clay modeling system. Proceedings of VRST '97: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, Lausanne, Switzerland, 197–200.
Arata, H. , Y. Takai , N. K. Takai , and T. Yamamoto . 1999. Free-form shape modeling by 3D cellular automata. Proceedings of SMI. International Conference on Shape Modeling and Applications, Aizu, Japan, 242–247.
Druon, S. , A. Crosnier , and L. Brigandat . 2003. Efficient cellular automata for 2D/3D free-form modeling. Proceedings of WSCG. 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2003, Plzen-Bory, Czech Republic.
Wu, P. , X. Wu , and G. A. Wainer . 2004. Applying cell-DEVS in 3D free-form shape modeling. Proceedings of Cellular Automata, 6th International Conference on Cellular Automata. ACRI 2004; Lecture Notes in Computer Science, Amsterdam, 81–90.

Models of Artificial Systems, Networking, and Communications

Stallings, W. 1996. Computer organization and architecture, 4th ed. New York: Macmillan.
Tanenbaum, A. 1990. Structured computer organization, 3rd ed. Upper Saddle River, NJ: Prentice Hall.
Hennessy, J. , and D. Patterson . 1994. Computer architecture: A quantitative approach. Upper Saddle River, NJ: Prentice Hall International.
Daicz, S. , A. Troccoli , and G. Wainer . 2001. Experiences in modeling and simulation of computer architectures using DEVS. Transactions of the Society for Modeling and Simulation International 18:179–202.
Daicz, S. , A. Troccoli , G. Wainer , and S. Zlotnik . 2000. Using the DEVS paradigm to implement a simulated processor. Proceedings of 33rd IEEE/SCS Annual Simulation Symposium, Washington, D.C.
Wainer, G. , S. S. Daicz , L. De Simoni , and D. Wasserman . 2001. Using the ALFA-1 simulated processor for educational purposes. ACM Journal on Educational Resources in Computing 1:111–151.
Ameghino, J. , and G. Wainer . 2000. Application of the cell-DEVS paradigm using N-CD++. Proceedings of 32nd Summer Computer Simulation Conference, Vancouver, Canada.
Behring, C. , M. Bracho , M. Castro , and J. A. Moreno . 2000. An algorithm for robot path planning with cellular automata. Proceedings of ACRI 2000, Karlsruhe, Germany.
Tzionas, P. , A. Thanailakis , and P. Tsalides . 1997. Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. IEEE Transactions on Robotics and Automation 13:237–246.
Wainer, G. 2006. Modeling robot path planning with CD++. Proceedings of ACRI 2006, LNCS 4173, Perpignan, France.
Butler, Z. , K. Kotay , D. Rus , and K. Tomita . 2002. Generic decentralized control for a class of self-reconfigurable robots. Proceedings of 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, Washington, D.C.
Narendra, K. S. , O. A. Driollet , M. Feiler , and K. George . 2003. Adaptive control using multiple models, switching and tuning. International Journal of Adaptive Control and Signal Processing 17:87–102.
Kofman, E. 2003. Quantized-state control. A method for discrete event control of continuous systems. Latin American Applied Research Journal 33:339–406.
Kofman, E. 2003. Discrete event control of time-varying plants. Technical report LSD0303, Universidad Nacional de Rosario, Argentina.
Campbell, A. 2005. Improvements to stochastic multiple model control: Hypothesis test switching and a modified model arrangement. MASc thesis, Carleton University, Ottawa, ON, Canada.

Campbell, A. , and G. Wainer . 2006. Applying DEVS modeling for discrete event multiple model control of a time varying plant. Proceedings of Winter Simulation Conference, Monterey, CA.
Hedrick, C. 1988. Routing Information Protocol. Network Working Group, Request for Comments 1058.
Altman, E. , and T. Jiménez . 2003. NS simulator for beginners. Technical report, INRIA Sophia-Antipolis. http://www-sop.inria.fr/mistral/personnel/Eitan.Altman/COURS-NS/n3.pdf
Bajaj, L. , M. Takai , R. Ahuja , K. Tang , R. Bagrodia , and M. Gerla . 1999. GloMoSim: A scalable network simulation environment. Technical report 990027, UCLA Computer Science Department.
Chang, X. 1999. Network Simulations with OPNET. Proceedings of the 31st Winter Simulation Conference, Phoenix, AZ.
Varga, A. 2001. The OMNeT++ discrete event simulation system. Proceedings of the European Simulation Multiconference, Prague, Czech Republic.
Tanenbaum, A. S. 2003. Computer networks. Upper Saddle River, NJ: Prentice Hall.
Gutowitz, H. 1995. Cellular automata and the sciences of complexity. Parts I–II. Complexity 1:16–22.
Wolfram, S. 2002. A new kind of science. Champaign, IL: Wolfram Media.
Ahmed, M. A. E. , K. Yonis , A. Elsahfei , and G. Wainer . 2005. Design and implementation of a library of network protocols in CD++. Proceedings of ANSS ’05: 38th Annual Simulation Symposium, Washington, D.C.
RFC-editor . 2003. Official Internet protocol standards. RFC 791. ftp://ftp.rfc-editor.org/in-notes/rfc791.txt
Malkin, G. 1998. RIP version 2. RFC 2453. Network Working Group, request for comments.
Farooq, U. , G. A. Wainer , and B. Balya . 2007. DEVS modeling of mobile wireless ad hoc networks. Simulation Modeling Practice and Theory 15:285–314.
Perkins, C. , E. Belding-Royer , and S. Das . 2003. Ad hoc on-demand distance vector (AODV) routing. IETF Network Working Group, RFC 3561.
Lee, C. Y. 1961. An algorithm for path connections and its applications. IRE Transactions on Electronic Computers EC-10, 2: 345–365.
Hochberger, C. , and R. Hoffmann . 1996. Solving routing problems with cellular automata. Proceedings of the Second Conference on Cellular Automata for Research and Industry, Milan, Italy.

Models of Urban Traffic

France, J. , and A. A. Ghorbani . 2003. A multiagent system for optimizing urban traffic. Proceedings of IAT ’03: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, Halifax, Canada.
Dresner, K. , and P. Stone . 2005. Multiagent traffic management: An improved intersection control mechanism. Proceedings of AAMAS ’05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecth University, the Netherlands, 471–477.
Schmidt, M. , R. Schäferr , and K. Nökel . 1998. SIMTRAP: Simulation of traffic-induced air pollution. Transactions of the Society for Computer Simulation International 15:122–132.
Treiber, M. , A. Hennecke , and D. Helbing . 2000. Congested traffic states in empirical observations and microscopic simulations. Physical Review E 62:1805.
Wagner, P. , K. Nagel , and D. Wolf . 1997. Realistic multi-line traffic rules for cellular automaton. Physica A 234:687.
Maniezzo, V. 2004. CA and roundabout traffic simulation. Proceedings of Sixth International Conference on Cellular Automata for Research and Industry, Amsterdam, the Netherlands, LNCS, vol. 3305.
Esser, J. , and M. Schreckenberg . 1997. Microscopic simulation of urban traffic based on cellular automata. International Journal of Modern Physics C 8:1025.
Marinosson, S. 2002. Simulation of the Autobahn traffic in North Rhine-Westphalia. Proceedings of 5th International Conference on Cellular Automata for Research and Industry, Geneva, Switzerland, LNCS, vol. 2493.
Rickert, M. , K. Nagel , M. Schreckenberg , and A. Latour . 1996. Two lane traffic simulations using cellular automata. Physica A 231: 44, 534–550.
Chi, S. , J. Lee , and Y. Kim . 1997. Using the SES/MB framework to analyze traffic flow. Transactions of the SCS 14 (4): 211–221.
Chou, H. , W. Huang , and J. A. Reggia . 2002. The trend cellular automata programming environment. Simulation 78:59–75.
Tolba, C. , D. Lefebvre , P. Thomas , and A. El Moudni . 2005. Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modeling. Simulation Modeling Practice and Theory 13:407–436.
Basile, F. , C. Carbone , P. Chiacchio , R. K. Boel , and C. C. Avram . 2004. A hybrid model for urban traffic control. Proceedings of 2004 IEEE International Conference on Systems, Man and Cybernetics, 1795–1800.
Kosonen, I. , and M. Pursula . 2007. HUTSIM. URL:http://www.tkk.fi/Units/Transportation/HUTSIM/ Accessed: 5/3/2007.
Owen, L. E. , Y. Zhang , L. Rao , and G. McHale . 2000. Street and traffic simulation: Traffic flow simulation using CORSIM. Proceedings of WSC ’00, 32nd Winter Simulation Conference, Orlando, FL, 1143–1147.
Chopard, B. , P. A. Queloz , and P. Luthi . 1996. Cellular automata model of car traffic in two-dimensional street networks. Journal of Physics A 29:2325–2336.
Barceló, J. , E. Codina , J. Casas , J. L. Ferrer , and D. García . 2005. Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems. Journal of Intelligent and Robotic Systems 41: 173–203, 01/01.
Cameron . 1996. PARAMICS—Parallel microscopic simulation of road traffic. Journal of Supercomputing 10:25.
Wainer, G. 1998. Discrete-event cellular models with explicit delays. PhD thesis, Université d’Aix-Marseille III, France.

Wainer, G. 2006. ATLAS: A language to specify traffic models using cell-DEVS. Simulation Modeling Practice and Theory 14:313–337.
Wainer, G. 2007. Defining a traffic modeling language using cellular discrete-event abstractions. Journal of Cellular Automata 2:291–343.
Wainer, G. 2007. Developing a software toolkit for urban traffic modeling. Software Practice and Experiment 37:1377–1404.
Diaz, A. , V. Vazquez , and G. Wainer . 2001. Application of the ATLAS language in models of urban traffic. Proceedings of the 34th Annual Simulation Symposium, Seattle, WA.
Tartaro, M. , C. Torres , and G. Wainer . 2001. Defining models of urban traffic using the TSC tool. Proceedings of Winter Simulation Conference, Washington, D.C.
Wainer, G. , S. Borho , and J. Pittner . 2001. Defining and visualizing models of urban traffic. Proceedings of 1st Mediterranean Multiconference on Modeling and Simulation, Genoa, Italy.

Building DEVS Simulators

Zeigler, B. P. , H. Praehofer , and T. G. Kim . 2000. Theory of modeling and simulation, 2nd. ed. New York: Academic Press.
Rodriguez, D. , and G. Wainer . 1999. New extensions to the CD++ tool. Proceedings of 31st SCS Summer Computer Simulation Conference, Chicago, IL.
Barylko, A. , J. Beyoglonián , and G. Wainer . 1998. GAD: A general application DEVS environment. Proceedings of Applied Modeling and Simulation, Honolulu, HI.
Fujimoto, R. M. 1999. Parallel and distribution simulation systems. New York: John Wiley & Sons.
Fujimoto, R. 1990. Parallel simulation of discrete events. Communications of the ACM 33 (10): 30–53.
Nicol, D. , and R. Fujimoto . 1994. Parallel simulation today. Annals of Operations Research 53:249–285.
Banks, J. , J. S. Carson , B. L. Nelson , and D. Nicol . 2005. Discrete-event system simulation, 4th ed. Upper Saddle River, NJ: Prentice Hall.
Chandy, K. , and J. Misra . 1981. Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM 24:198–206.
Chandy, K. , and J. Misra . 1979. Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering SE-5:440–452.
Bryant, R. E. 1977. Simulation of packet communication architecture computer systems. Technical report, UMI order number: TR-188. Massachusetts Institute of Technology.
Jefferson, D. 1987. Distributed simulation and the time warp operating system. Proceedings of 11th Symposium on Operating Systems Principles, Austin, TX.
Jefferson, D. 1985. Virtual time. ACM Transactions on Programming Languages and Systems 7:404–425.
Henning, M. , and S. Vinoski . 1999. Advanced CORBA programming with C++. Reading, MA: Addison–Wesley.
Dongarra, J. J. 1995. High-performance computing: Technology, methods and applications. Amsterdam: Elsevier.
Sunderam, V. , A. Geist , J. Dongarra , and R. Manchek . 1994. The PVM concurrent computing system: Evolution, experience and trends. Parallel Computing 20:531–545.
Alonso, G. 2003. Web Services: Concepts, architectures and applications. New York: Springer–Verlag.
IEEE Std 1278 . 1995. IEEE standard for modeling and simulation. Distributed interactive simulation (DIS).
IEEE Std 1516.1-2000 . 2001. IEEE standard for modeling and simulation. High-level architecture (HLA)—Federate interface specification, i–467.
Cho, Y. W. , X. Hu , and B. Zeigler . 2003. The RTDEVS/CORBA environment for simulation-based design of distributed real-time systems. Simulation 79 (4): 197–210.
Sarjoughian, H. S. , and B. P. Zeigler . 2000. DEVS and HLA: Complementary paradigms for M&S? Transactions of the SCS 17:187–197.
Kim, K. , and W. Kang . 2004. CORBA-based, multithreaded distributed simulation of hierarchical DEVS models: Transforming model structure into a nonhierarchical one. Proceedings of ICCSA 2004, International Conference, Assisi, Italy, 167–176.
Seo, C. , S. Park , B. Kim , S. Cheon , and B. P. Zeigler . 2004. Implementation of distributed high-performance DEVS simulation framework in the grid computing environment. Proceedings of High Performance Computing Symposium, Advanced Simulation Technology Conference, Arlington, VA.
Cheon, S. , C. Seo , S. Park , and B. P. Zeigler . 2004. Design and implementation of distributed DEVS simulation in a peer-to-peer network system. Proceedings of DASD, Advanced Simulation Technology Conference, Arlington, VA.
Chow, A. C. , and B. Zeigler . 1994. Parallel DEVS: A parallel, hierarchical, modular modeling formalism. Proceedings of Winter Simulation Conference, Orlando, FL.
Chow, A. C. , D. H. Kim , and B. P. Zeigler . 1994. Abstract simulator for the P-DEVS formalism. Proceedings of AI, Simulation, and Planning in High Autonomy Systems, Gainesville, FL.
Kim, K. H. , Y. R. Seong , T. G. Kim , and K. H. Park . 1996. Distributed simulation of hierarchical DEVS models: Hierarchical scheduling locally and time warp globally. Transactions of the SCS 13 (3): 135–154.
Glinsky, E. , and G. Wainer . 2006. New parallel simulation techniques of DEVS and cell-DEVS in CD++. Proceedings of Annual Simulation Symposium, Huntsville, AL, 244–251.

Glinsky, E. , and G. Wainer . 2002. Performance analysis of real-time DEVS models. Proceedings of Winter Simulation Conference, San Diego, CA.
Glinsky, E. , and G. Wainer . 2002. Performance analysis of DEVS environments. Proceedings of Artificial Intelligence, Simulation and Planning. Lisbon, Portugal.
Kim, K. , W. Kang , B. Sagong , and H. Seo . 2000. Efficient distributed simulation of hierarchical DEVS models: Transforming model structure into a nonhierarchical one. Proceedings of 33rd Annual Simulation Symposium, Washington, D.C.
Cho, S. , and T. G. Kim . 2001. Real-time simulation framework for RT-DEVS models. Transactions of the Society for Computer Simulation International 18:203–215.
Wainer, G. , and N. Giambiasi . 2001. Application of the cell-DEVS paradigm for cell spaces modeling and simulation. Simulation 76 (1): 22–39.
Wainer, G. 1998. Discrete-event cellular models with explicit delays. PhD thesis, Université d'AixMarseille III, France.
Troccoli, A. , and G. Wainer . 2003. Implementing parallel cell-DEVS. Proceedings of 36th IEEE/SCS Annual Simulation Symposium, Orlando, FL.
Madhoun, R. , B. Feng , and G. Wainer . 2007. Web-service-based distributed CD++. Proceedings of AIS 2007, Artificial Intelligence, Simulation and Planning, Buenos Aires, Argentina.
Schulz, S. , J. W. Rozenblit , M. Mrva , and K. Buchenriede . 1998. Model-based codesign. Computer 31:60–67.
Hong, J. , H. Song , T. G. Kim , and K. H. Park . 1997. A real-time discrete event system specification formalism for seamless real-time software development. Discrete Event Dynamic Systems: Theory and Applications 7:355–375.
Kim, T. G. , S. M. Cho , and W. B. Lee . 2000. DEVS framework for systems development: Unified specification for logical analysis, performance evaluation and implementation. In Discrete event modeling & simulation: Enabling future technologies, ed. H. S. Sarjoughian and F. Cellier . New York: Springer–Verlag.
Glinsky, E. , and G. Wainer . 2002. Definition of RT simulation in the CD++ toolkit. Proceedings of Summer Computer Simulation Conference, San Diego, CA.
Li, L. , T. Pearce , and G. Wainer . 2002. An experience in hardware–software codesign using the DEVS formalism. Proceedings of EuroSim Industrial Simulation Symposium 2003, Valencia, Spain.
Glinsky, E. , and G. Wainer . 2004. Modeling and simulation of systems with hardware-in-the-loop. Proceedings of Winter Simulation Conference, Washington, D.C.
Jacques, C. , and G. Wainer . 2002. Using the CD++ DEVS toolkit to develop Petri nets. Proceedings of Summer Computer Simulation Conference, San Diego, CA.
Glinsky E., and G. Wainer . 2004. Model-based development of embedded systems with RT-CD++. Proceedings of the WIP Session, IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, ON, Canada.
Wainer, G. , E. Glinsky , and P. MacSween , 2005. Model-driven architecture of real-time systems. In Model-driven software development—Research and practice in software engineering, vol. II, ed. S. Beydeda and V. Gruhn . New York: Springer–Verlag.
Yu, H. , and G. Wainer . 2007. E-CD++: An environment for developing embedded DEVS applications. Carleton University, Dept. of Systems and Computer Engineering.
Barros, F. J. 1995. Dynamic structure discrete event system specifications: A new formalism for dynamic structure modeling and simulation. Proceedings of Winter Simulation Conference, Arlington, VA, 781–785.
Barros, F. 1998. Abstract simulators for the DSDE formalism. Proceedings of Winter Simulation Conference, Washington, D.C., 407–412.
Uhrmacher, A. M. 2001. Dynamic structure in modeling and simulation: A reflective approach. ACM Transactions on Modeling and Computer Simulation 11:206–232.
Uhrmacher, A. M. , and J. Himmeelspach . 2004. Processing dynamic PDEVS models. Proceedings of 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), Volenlam, the Netherlands.
Shang, H. , and G. Wainer . 2006. A simulation algorithm for dynamic structure DEVS modeling. Proceedings of Winter Simulation Conference, Monterey, CA.
Shang, H. , and G. Wainer . 2008. Dynamic structure DEVS: Improving the real-time embedded systems simulation and design. Proceedings of 40th IEEE/SCS Annual Simulation Symposium, Ottawa, Canada.
Kgwadi, M. , H. Shang , and G. Wainer . 2008. Definition of dynamic DEVS models—Dynamic structure CD++. Proceedings of Poster Papers Workshop, SpringSim 2008, Ottawa, Canada.
Christensen, E. , F. Curbera , G. Meredith , and S. Weerawarana . 2006. Web Service description language (WSDL) 1.1. URL: http://www.w3.org/TR/wsdl
Gudgin, M. , M. Hadley , N. Mendelsohn , J. Moreau , and H. Nielsen . 2006. SOAP version 1.2 part 1: Messaging framework. URL: http://www.w3.org/TR/soap12-part1/
Madhoun, R. , and G. Wainer . 2007. Performance analysis of Web-based CD++. Presented at DEVS Symposium 2007, Norfolk, VA,
Wainer, G. , Q. Liu , J. Landais , L. Quinet , and M. K. Traoré . 2008. Performance analysis of Web-based distributed simulation in DCD++: A case study across the Atlantic Ocean. Presented at SCS High Performance Computing and Simulation Symposium (HPCS 2008), Ottawa, Canada.
Lombardi, S. , G. Wainer , and B. P. Zeigler . 2006. Interoperation of DEVS models in DEVS/C# and CD++. Proceedings of SISO Fall Interoperability Workshop, Huntsville, AL.

Mechanisms for Three-Dimensional Visualization

Choi, W. 2008. Study on L-V-C (live-virtual-constructive) interoperation for the national defense M&S (modeling & simulation). ICISS International Conference on Information Science and Security, Seoul, Korea, 128–133.

Ames, A. , D. Nadeau , and J. Moreland . 1997. VRML 2.0 source, 2nd ed. New York: John Wiley & Sons, Inc.

Pardew, L. , and M. Tidwell . 2006. Autodesk Maya and Autodesk 3ds Max side-by-side. Boston: Course Technology Press.

Segal, M. , and K. Akeley . 2006. The OpenGL graphics system: A specification. Silicon Graphics, Inc.

Roosendaal, T. , and S. Selleri . 2004. The official Blender 2.3 guide: Free 3D creation suite for modeling, animation, and rendering. San Francisco, CA: No Starch Press.

Behr, J. , P. Dähne , and M. Roth . 2004. Utilizing X3D for immersive environments. Proceedings of Web3D ’04, Ninth International Conference on 3D Web Technology, Monterey, CA, 71–78.

Roehl, B. , and J. Couch . 1997. Late night VRML 2.0 with Java. Hightstown, NJ: Ziff–Davis Publishing Co.

Wainer, G. , and W. Chen . 2003. A framework for remote execution and visualization of cell-DEVS models. Simulation 79:626–647.

Wainer, G. , S. Borho , and J. Pittner . 2001. Defining and visualizing models of urban traffic. Proceedings of 1st Mediterranean Multiconference on Modeling and Simulation, Genoa, Italy.

Sourceforge . 2007. JHotDraw Open-Source Project. URL: http://sourceforge.net/projects/jhotdraw

Wainer, G. 2007. Developing a software toolkit for urban traffic modeling. Software Practice and Experience 37:1377–1404.

Khan, A. , and G. Wainer . 2005. Advanced visualization of DEVS and cell-DEVS models in Maya. Proceedings of the SISO Spring Interoperability Workshop, San Diego, CA.

Djafarzadeh, R. , T. Mussivand , and G. Wainer . 2005. Modeling energy pathways in cells. Proceedings of 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference, San Diego, CA.

Wenhola, W. , and G. Wainer . 2006. DEVSView: A tool for visualizing CD++ simulation models. Proceedings of DEVS Symposium, Spring Simulation Conference, Huntsville, AL.

Kilgard, M. J. 1996. The OpenGL utility toolkit (GLUT) programming interface: API Version 3. Available: http://www.Opengl.org/resources/libraries/glut/glut-3.spec.pdf

Jackins, C. L. , and S. L. Tanimoto . 1980. Oct-trees and their use in representing three-dimensional objects. Computer Graphics & Image Processing 14:249–270.