ENHANCING HUMAN PERFORMANCE VIA SIMULATION-BASED TRAINING AND AIDING
MODELING AND SIMULATIONS FOR LEARNING AND INSTRUCTION
Volume 3

Series Editors
J. Michael Spector
Learning Systems Institute, Florida State University, Tallahassee, USA
Norbert M. Seel
University of Freiburg, Germany and Florida State University, Tallahassee, USA
Konrad Morgan
Human Computer Interaction, University of Bergen, Norway

Scope
Models and simulations have become part and parcel of advanced learning environments, performance technologies and knowledge management systems. This book series will address the nature and types of models and simulations from multiple perspectives and in a variety of contexts in order to provide a foundation for their effective integration into teaching and learning. While much has been written about models and simulations, little has been written about the underlying instructional design principles and the varieties of ways for effective use of models and simulations in learning and instruction. This book series will provide a practical guide for designing and using models and simulations to support learning and to enhance performance and it will provide a comprehensive framework for conducting research on educational uses of models and simulations.

A unifying thread of this series is a view of models and simulations as learning and instructional objects. Conceptual and mathematical models and their uses will be described. Examples of different types of simulations, including discrete event and continuous process simulations, will be elaborated in various contexts. A rationale and methodology for the design of interactive models and simulations will be presented, along with a variety of uses ranging from assessment tools to simulation games. The key role of models and simulations in knowledge construction and representation will be described, and a rationale and strategy for their integration into knowledge management and performance support systems will provided.

Audience
The primary audience for this book series will be educators, developers and researchers involved in the design, implementation, use and evaluation of models and simulations to support learning and instruction. Instructors and students in educational technology, instructional research and technology-based learning will benefit from this series.
Enhancing Human Performance Via Simulation-Based Training and Aiding

A Guide to Design and Development

By

Douglas M. Towne
University of Southern California, St Helena, CA, USA
A C.I.P. record for this book is available from the Library of Congress.


Published by: Sense Publishers,
P.O. Box 21858, 3001 AW Rotterdam, The Netherlands
http://www.sensepublishers.com

Printed on acid-free paper

All rights reserved © 2007 Sense Publishers

No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.
# TABLE OF CONTENTS

PREFACE...................................................................................................................................... VII

ACKNOWLEDGEMENTS................................................................................................................ IX

CHAPTER 1. INTRODUCTION ............................................................................................... 1
   SCOPE OF THIS VOLUME ........................................................................................................ 1
   SYSTEM OVERVIEW ............................................................................................................. 2
   OBJECT ORIENTED DESIGN ............................................................................................ 4
   ORGANIZATION AND CONTENT OF THE VOLUME ......................................................... 8

CHAPTER 2. RESOURCES FOR SIMULATION DEVELOPMENT .......... 9
   ADOBE FLASH MX.............................................................................................................. 9
   ALTERNATE SIMULATION METHODOLOGIES ............................................................. 20
   INTRODUCTION TO REACT ........................................................................................... 29

CHAPTER 3. DESIGN OF SIMULATION COMPONENTS ................. 33
   GENERAL ISSUES .............................................................................................................. 33
   CHARACTERISTICS OF REACT COMPONENTS ............................................................ 37
   THE BASE CLASS SIMULATION OBJECT ................................................................ 38
   THE TWO BASIC COMPONENT TYPES: CONTROLS AND REACTORS ..................... 43
   THE CONTROL COMPONENT TYPE ............................................................................. 43
   THE REACTOR COMPONENT TYPE ............................................................................. 54

CHAPTER 4. MODEL BUILDING ........................................................................... 65
   A FIVE MINUTE MODEL ................................................................................................. 65
   A REAL FRONT PANEL ................................................................................................... 68
   SIMULATING UNSEEN FUNCTIONS .............................................................................. 71
   SIMULATING A SYSTEM WITH INTERNAL MOVING PARTS ...................................... 73
   MODULARIZING LARGE MODELS .............................................................................. 80
   MODELS WITH VIRTUAL AGENTS AND UNKNOWNS .............................................. 80
   ELEMENTS OF THE CIC USER INTERFACE ................................................................. 82

CHAPTER 5. SIMULATING REAL-TIME PROCESSES ..................... 87
   ANIMATION METHODS ................................................................................................. 87
   A FLEXIBLE FUNCTION FOR CHANGING PROPERTIES OVER TIME ...................... 90
   CONTINUOUS MONITORING OF CONDITIONS ......................................................... 95
   PERIODIC MODEL UPDATING ...................................................................................... 97
   THE CIC SIMULATION ................................................................................................. 97

CHAPTER 6. DEMONSTRATING BEHAVIORS AND CONCEPTS ....... 99
   USE OF SIMULATIONS IN THE CLASSROOM............................................................. 99

v
TABLE OF CONTENTS

DEMONSTRATION OF BASIC CONCEPTS .............................................................. 103
DEMONSTRATING AND EXPLICATING COMPLEX SUBJECT MATTER .............. 107
RECAP OF DEMONSTRATION METHODS ............................................................. 114

CHAPTER 7. MODEL-BASED INSTRUCTION .......................................................... 115
  STRUCTURED EXERCISES FOR INCREASING AND ASSESSING PROFICIENCY ... 115
  MANAGING INSTRUCTIONAL DELIVERY .............................................................. 127
  RANGE OF APPLICATIONS OF STRUCTURED METHODS .............................. 130
  THE STRUCTURED METHOD WITH CUSTOMIZED ITEMS ............................... 135

CHAPTER 8. INSTRUCTING FAULT DIAGNOSIS ............................................... 139
  DOMAIN-INDEPENDENT INSTRUCTION OF FAULT DIAGNOSIS ....................... 140
  DOMAIN SPECIFIC INSTRUCTION OF FAULT DIAGNOSIS ............................. 140
  PART TASK INSTRUCTION OF DIAGNOSTIC SKILLS AND KNOWLEDGE ......... 141
  INTELLIGENT GUIDANCE OF TROUBLESHOOTING PRACTICE ....................... 146
  APPLICATIONS .................................................................................................. 159

CHAPTER 9. SCENARIO-BASED INSTRUCTION .................................................... 161
  DYNAMIC, DISCRETE-ACTION TASKS ................................................................. 161
  APPLICATION ISSUES .......................................................................................... 163
  SYSTEM OVERVIEW .......................................................................................... 165
  APPLICATION DEVELOPMENT ............................................................................ 166
  INSTRUCTIONAL DELIVERY ............................................................................... 182
  SUMMARY AND CONCLUSIONS ........................................................................... 186

CHAPTER 10. SIMULATION-BASED PERFORMANCE SUPPORT ................................................................................................................. 191
  SUPPORT OF DIAGNOSTIC REASONING TASKS .............................................. 191
  AIDING PERFORMANCE OF COMPLEX PROCEDURES .................................... 196
  AIDING SYSTEM-LEVEL DIAGNOSTIC PERFORMANCE ................................. 206

CHAPTER 11. INTERACTIVE TECHNICAL DOCUMENTATION .................................. 213
  SYSTEM OVERVIEW .......................................................................................... 213
  IETM DEVELOPMENT ....................................................................................... 215
  USING A PKS IETM ........................................................................................... 220
  RANGE OF APPLICATION ................................................................................... 228
  SUMMARY OF PKS IETM APPLICATIONS .......................................................... 230

CHAPTER 12. SIMULATIONS IN MULTIPLE USES .................................................. 231
  A COMPLETE TECHNICAL SUPPORT APPLICATION .................................... 231
  SUMMARY OF MULTIPLE-USE APPLICATIONS ............................................... 238

REFERENCES ..................................................................................................... 239

INDEX .................................................................................................................. 243

GLOSSARY ........................................................................................................... 247

VI
PREFACE

This volume presents methodologies, designs, and applications developed over the past ten years in the pursuit of effective creation and use of simulations to support technical people in understanding, operating, and maintaining complex systems. While it does not intend to represent a survey of all the pertinent research in simulation-based training and performance support, citations are provided for the key research that either influenced the work or relates to it in some direct way.

While the focus of this volume is upon the design and function of development and delivery systems, some snippets of program code are sprinkled through the early chapters, in an effort to reflect the relative ease with which key processes are implemented when the design anticipates the needs. The workings of these code snippets are fully described in the text as well, so that those less intrigued with such matters can fully follow the discussions.

On-line Examples

Many of the sample applications discussed here can be found and operated at the following Web site: http://www-rcf.usc.edu/~dtowne/.

Readers are encouraged to view and execute these on-line samples, since the static and monochromatic figures and accompanying text presented here cannot fully represent all the interactions to be experienced there.
ACKNOWLEDGEMENTS

The majority of this work was supported by the Office of Naval Research (ONR), much of it administered by ONR and in later years by Naval Air Warfare Center, Training Systems Division (NAWCTSD), Orlando.

We are indebted to ONR for their support over the years, as well as other government funding agencies that supported earlier work, including USAF Armstrong Laboratories, Texas, and the Navy Personnel Research and Development Center (NPRDC), San Diego, CA.

We thank the many Navy technicians, training specialists, and site commanders who contributed their time and expertise to assisting us in developing and evaluating training and performance aiding systems in their midst.

The work in training fighting of high rise fires was made possible by the generous participation of members of the San Francisco Fire Department, the Los Angeles City Fire Department, and the Del Amo Financial Center, Torrance, CA. Quentin Pizzini and Donna Darling, of our organization, developed key parts of the training application.

We thank Merle Vogel (CSCWP, San Diego) for his substantial contributions to and support of the simulation of the aircraft nose wheel positioning system.

We thank the individuals, all of whom are cited by name in the various research reports referenced in this volume, without whose able contributions the practical applications could not have been produced and evaluated.
CHAPTER 1

INTRODUCTION

In recent years the tools and technology to produce and deliver realistic simulation-based systems for enhancing proficiency have become widely available at very reasonable costs. These proficiency enhancement resources include instructional systems, supportive practice environments, collaborative systems for aiding job performance, and interactive technical documentation. While it might be argued that resources that assist or support the individual in performing a task do not truly enhance proficiency, we consider such approaches as legitimate members of the family of methods that can improve the ultimate quality of job performance, and we include them in this volume.

Experience in employing these tools clearly indicates that effective applications can be produced at very reasonable costs, particularly when the simulation of the target system or problem environment is reused to serve more than one of the alternatives for improving performance.

SCOPE OF THIS VOLUME

Some of the most critical questions that face the decision maker considering embarking on a simulation development program for training and/or performance aiding are:

– What sort of simulation methodology should be employed?
– What development tools and programming language are most appropriate and useful for this purpose?
– How may instructional content and processes be produced to work in association with simulation resources?
– How may simulation products be used to support both training and performance aiding requirements?
– What skills and previous experience are required to produce the simulation resources and the instructional application?

This volume will attempt to directly address these questions with detailed accounts of systems developed and applications produced.

Terminology

The terms model and simulation will be used extensively and somewhat interchangeably throughout, however we will usually reserve model to refer to one or more emulations of identifiable hardware systems, while simulation is used
when discussing the emulation of a task environment consisting of system models as well as user interfaces for performing a task. In either case, the emphasis is on emulating the real world to the level of realism required to support learning and performance, rather than accomplishing engineering analysis or system design. While the model building methods presented here are certainly up to the task of creating such predictive simulations, the effort to do so is typically unwarranted when producing instructional and performance aiding applications.

SYSTEM OVERVIEW

The family of development and delivery systems that will be covered in this volume is shown in Figure 1-1.
The key elements of this architecture, starting from the bottom of the figure, are: 1) a simulation development system in which models are produced; 2) one or more models or simulations that emulate some real system or task environment; and 3) a number of domain-independent software systems, each designed to utilize and operate upon the domain-specific system models to deliver a particular kind of training or performance aiding activity.

Upon producing a model of a particular system or task environment, the developer interacts with one or more of the development systems to produce the particular types of training and/or performance aiding application desired. While one would rarely employ all seven systems to a particular domain or task, it is likely that two or more of the development systems would be used to produce complementary instructional, performance aiding, or technical support products for the domain. The final chapter presents an application involving five of the seven development systems shown.

The Central Design Concept

A key concept that the figure cannot adequately convey is that the domain-specific models remain as intact entities that serve as the representation of the real world when the various training and aiding systems interact with a learner/performer. With just one exception – scenario-based training – the models are not somehow customized by the developer to support the particular training or performance aiding application they will serve. Instead, the models are called upon and acted upon by the delivery systems in various ways that are specific to the particular training or aiding function being carried out.

This is not to say that during model development we ignore the ultimate training and aiding processes we wish to carry out, but that the design of the basic elements of models is made sufficiently robust that the delivery systems have all the necessary control over the models to accomplish their functions.

In a similar fashion, the training and aiding processes are not modified to deal with specific domain models. This approach to modularizing the content separately from the delivery processes is just one of the ways in which the methodology described here employs object oriented design, which will be elaborated below.

The Exception: Scenario-based Training

In the case of scenario-based training, discussed in Chapter 9, the simulation of the task environment is typically developed specifically to support that training requirement, and the simulation necessarily maintains some training-specific variables that reflect the proficiency with which the learner has performed various functions. Even in this case, however, the simulation of the problem environment could be used to serve another purpose, such as performance aiding, with little modification.
CHAPTER 1

OBJECT ORIENTED DESIGN

The principles of object oriented programming (OOP) and object oriented design (OOD) are central to all the systems and methods discussed in this volume, and the remainder of this chapter will briefly outline how these principles apply to the world of simulation-based instruction and simulation-enabled performance support. In the words of Booch (1991), object oriented design is

… built upon a sound engineering foundation, whose elements we collectively call the object model. The object model encompasses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence. By themselves, none of these principles are new. What is important about the object model is that these elements are brought together in a synergistic way.

When these principles are applied to model development and instructional system development in a principled manner, the result is a system that offers increased transparency to inspection, ease of development and modification, and flexibility in reuse of elements.

These principles of object oriented design, or OOD, apply both to the models that are produced to support training and performance aiding and to the design of the software systems that rely upon those models to accomplish their objectives. Since entire volumes are required to even scratch the surface of all that OOD embodies, this section will only briefly consider the key elements of OOD and how they are evidenced in the system architecture that is the subject of this volume.

Abstraction

Abstraction refers to the process of representing more complex entities and systems in ways that reduce the apparent complexity through simplification of the representation. One objective of abstraction is to ease understanding by retaining the essential characteristics and processes that underlie the complex subject while omitting, combining, or further simplifying those aspects that can be so treated without materially altering the character of the system at the level being developed for use.

Simulations are, by definition, abstractions, for regardless of the level of detail presented, there are almost always more detailed ways to represent the elements that constitute the model. Even a highly detailed simulation of an electrical circuit involves abstraction, unless it happens to model the deepest known electrical phenomena at the quantum level, which is probably impossible.

When the intent of a simulation is to serve training and aiding purposes, the model is made as simple as will support the objectives of the application. Thus, if a model is constructed to teach a procedure on a front panel, there is no benefit derived from modeling the unseen elements that produce the effects that appear at the panel beyond that necessary to support the correct front panel responses. Instead, the observable behavior of the front panel elements can be expressed in
terms of the states of the other observable elements. This is not to say that some functions of internal elements are never expressed in the model, for such functionality may be the essence of the transformation that occurs between the controls that are manipulated and the indicators that reflect change, but typically those internal functions can be highly simplified while still reproducing the correct system behavior.

A model of a device produced to support training in fault diagnosis, on the other hand, must involve some level of functional representation of the components of that system, so that faults can be produced and represented in an accurate fashion. These functional elements, however, can themselves often be massively simplified, so that the learner observes their normal and abnormal behaviors rather than the deep physical phenomena that produce those behaviors.

**Encapsulation**

While abstraction refers to the extent to which a model element is simplified, encapsulation refers to the process of hiding the implementation of that abstraction so that neither the developer nor other training/aiding software systems need be concerned with those details (except, of course, the original developer of the element). Once a model entity is constructed, and the abstraction of its function is encapsulated, the developer needs not again be concerned with how that functionality was produced.

As we shall see shortly, a key issue in the design of a simulation system is the manner in which domain-independent behaviors of model elements are separated from their domain-dependent behaviors, so that the former can be encapsulated and the latter can be specified each time the element is employed in a particular model.

A key aspect of the simulation development process that will be covered in Chapter 2 involves encapsulating the internal operation of model elements into well-defined functions with predefined names. Under the design of the ReAct simulation system, the function `setState` handles the graphical updating task for all model objects, regardless of type. Thus, once a particular type of control, for example, is constructed with a correct `setState` function, no future developer need spend time producing graphical updating functions when applying the object in a particular application. Of equal or more importance, other software systems, such as an instructional management system, can communicate with elements of models without having to be concerned with how the elements function. Thus those training and aiding systems can remain domain-independent rather than having to be customized to serve up the domain.

**Modularity**

Modularity is another possible manner of reducing complexity of a system representation, achieved by partitioning the overall system into individual elements that carry out functions that can be stated or described in a convenient way. Beginning programmers often produce programs composed of long sequences of
statements that reflect little of the structure of the problem at hand, even though the program might execute correctly. More advanced programmers tend to break down complex problems into modules (functions), each tackling a relatively concise sub-problem or represent an entity of a relatively narrow scope of operation.

Like abstraction, modularity enters into nearly all levels of simulations constructed to support training and performance aiding. At the lowest level of a system representation, it is both convenient and instructionally powerful to produce functional modules that carry out relatively simple functions that need not be represented or explained more deeply. These functional entities typically are physical entities in the real system, as well.

At higher levels in the system representation, a developer may, if the development environment permits, elect to produce a number of stand-alone simulations each handling some relatively well-defined and significant functions. Thus, a model of a radar system might be constructed of separate simulation modules representing the transmitter, the receiver, the signal processor, the display, and so on.

Finally, in the system architecture that will be outlined in this volume, there is significant modularity reflected in the very highest levels of system design. One module in a total training system is the simulation, another is the pedagogical content, and another is the training management system, and so on. By so separating the system model from the instructional content and the instructional management, the design permits reuse of the simulation to meet other purposes, and it greatly facilitates the division of labor during development and ease of maintenance when modifying or extending the application.

Hierarchy

As noted above, abstraction reduces the complexity of a system representation and encapsulation hides the internal details of the abstractions so that those details can essentially be ignored, once specified. Another opportunity for reducing the apparent complexity of a modeled system is to structure the representation hierarchically, thereby providing a structure for viewing the system at differing levels depending upon the need.

For example, by viewing an automobile as consisting of just a few basic functions, such as motive, braking, cooling, etc., the device seems more palatable and can be regarded as relatively simple. If the representation ended there, of course, the utility of the model would be highly limited. If, on the other hand, each subsystem is composed of another manageable layer of subsystems, and this decomposition continues to whatever level is required for the application, then nothing is lost by having grouped elements into families, and a great deal is gained.

The hierarchical structure of the automobile example is termed an object structure, or “part of” structure, in which a group of entities are part of another entity, or level. This means of viewing complex systems is so powerful it is employed in many complex example models given in this volume.
INTRODUCTION

There is a second kind of hierarchy, called the class structure, or “kind of” structure, in which one entity is a kind of another entity. For example, we could group all operable controls into a group, or class, and say that they are all of one kind, since they all respond directly to user actions upon them. This exact type of hierarchical grouping is employed in the object model discussed in Chapter 2.

Typing

The concept of type can be applied at very low levels of programming to define and restrict how different elements of data are to be interpreted and operated upon. At higher levels, and particularly in the context of simulation development, type plays an important role in distinguishing and specifying different kinds of model elements in a ways that permit processes to operate upon particular instances in useful ways since their type is warranted to be in conformance with some specification.

The discussions of Chapter 3, dealing with the development of reusable components, center heavily upon distinguishing among different kinds of model elements in generalized ways, which sets the stage for developing a hierarchy of model element types.

Concurrency

Many simple device models involve apparently instantaneous changes in the states of elements. Switches are flipped from on to off with no apparent intermediate states, and lights change from red to green without perceivable delay or process time. In reality, all change involves the passage of time, but we customarily ignore those short transition times that are of little consequence to understanding, and we often artificially shorten longer transition times that need not be experienced in real time.

In many real world systems, however, there are not only continuous changes that occur over time in ways that should be represented accurately, there are also changes that occur concurrently. The simulation development system presented in the next few chapters, particularly in Chapter 5, provides methods especially produced for representing real-time processes either singly or concurrently. Importantly, the approach supports true simulation of multiple processes, in which other objects may respond to continuous changes, as opposed to multiple animations that do affect other elements of the model.

Persistence

The last property of OOD discussed here is persistence, the preservation of object and system states from one time to another. Persistence of state is the assumed condition when running a single model, i.e., one expects that a change to object states will persist during a session. This does not necessarily mean that objects will not subsequently change from the state the user produced, as a result of some other
cause or even the passage of time, but simply that a change of viewpoint to another section of a large model will not by itself cause objects to revert to some initialized condition, unless that is an explicit wish of the developer.

Persistence becomes an issue when large models composed of individual simulations are executed. Again, one expects that operations performed on any of the individual subsystems will persist even if the particular simulation is unused and out of view for an extended period of time, during a session. And, as a special case, a developer would hope that a development system would provide some means for recording and recovering the state of a complex system when a very complex simulation is resumed after an interruption. This capability is also essential when a training system wishes to set up the simulation for some instructional purpose, such as replaying a completed exercise or demonstrating an expert procedure.

Persistence is explicitly addressed by the developer in producing interactive technical documentation, as covered in Chapter 11, and the development system provides options for either maintaining a system state from one viewing to another or reinitializing it with each viewing. For example, suppose we wish to display a device model in a particular state when a particular “page” of a document (or screen) is viewed. If the accompanying text says “… here is system ABC in mode XYZ”, then either the model should be made to not respond to user actions, which is possible, or it should be reinitialized to the stated mode each time that screen is viewed in a session.

ORGANIZATION AND CONTENT OF THE VOLUME

Figure 1-1 also reflects the general organization of this volume, working from bottom to top. Chapter 2 will provide an overview of the basic Adobe Flash development system and the simulation development resource (ReAct) that works in conjunction with Flash to facilitate model development. Chapters 3 through 5 will focus upon the simulation development process, and they provide a number of example products of that process. Chapters 6 through 9 will detail the design and implementation of the training resources shown, viz., demonstration of system functions and basic concepts; instruction of system structure, functions and operation; intelligent diagnostic training; and scenario-based training. Chapter 10 will do the same for procedural and diagnostic performance support, and Chapter 11 addresses interactive technical documentation. The volume concludes in Chapter 12 with a brief example of a complete integrated system that provides training, aiding, and interactive technical documentation for a particular domain.
CHAPTER 2

RESOURCES FOR SIMULATION DEVELOPMENT

There are a number of commercially available development environments, each with their own programming language that could be used for simulation development and execution. These include MS Visual Basic, .NET, ToolBook, various Java development systems, and the Adobe Flash system. Of these, Adobe Flash was selected as the preferable approach for reasons that include these:

– Its graphics are vector based and can be easily scaled and rotated.
– It provides very high programmatic control over graphic elements.
– Its programming language, ActionScript, is ECMA-standard.
– Many users worldwide produce products, training, and support.
– Its no-cost run-time system is well integrated into most Web browsers.
– Its applications are compiled and execute on both PCs and Mac platforms.
– Its applications can be deployed via Internet, without modification.

All of the simulation applications and training and aiding delivery systems to be discussed are developed in Adobe Flash augmented with a set of Flash-based simulation development functions termed ReAct. This chapter will outline the capabilities of these two systems with particular emphasis upon the manner in which they adhere to the principles of object oriented design. The development resources added by the ReAct system will only be introduced in this chapter, and will be discussed in considerably more detail in later chapters, as various simulation development issues are addressed.

ADOBE FLASH MX

Adobe Flash MX, hereafter termed Flash, is used worldwide by a vast number of developers to produce a wide range of multimedia applications. Primary among those applications are highly animated Web sites, the area in which Flash became a world leader. The system, however, is also fully capable of producing very sophisticated stand alone applications. Simulations can be developed and executed on both Windows-based PCs and Apple Macintosh platforms, ranging from tablet computers to desktops, and distributed applications require no modifications.

Importantly, Flash employs vector graphics as its graphical format, in which figures are composed of line segments. The great advantage of the vector graphic format is that figures can be scaled and rotated with virtually no loss in resolution or clarity, whereas the alternative graphical format, raster graphics, produces serious artifacts when scaled. Figure 2-1 illustrates the differences between a raster graphic and a vector graphic, both scaled to 200% from an original graphic.