SANDIA REPORT

SAND2001-0828

Unlimited Release

Printed March 2001

OBEST: The Object-Based Event Scenario Tree Methodology

Gregory D. Wyss and Felicia A. Durán

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.doe.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: <http://www.ntis.gov/ordering.htm>

OBEST: The Object-Based Event Scenario Tree Methodology

Gregory D. Wyss and Felicia A. Durán
Risk, Reliability & Modeling Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0747

Abstract

Event tree analysis and Monte Carlo-based discrete event simulation have been used in risk assessment studies for many years. This report details how features of these two methods can be combined with concepts from object-oriented analysis to develop a new risk assessment methodology with some of the best features of each. The resultant Object-Based Event Scenario Tree (OBEST) methodology enables an analyst to rapidly construct realistic models for scenarios for which an *a priori* discovery of event ordering is either cumbersome or impossible (especially those that exhibit inconsistent or variable event ordering, which are difficult to represent in an event tree analysis). Each scenario produced by OBEST is automatically associated with a likelihood estimate because probabilistic branching is integral to the object model definition. The OBEST method uses a recursive algorithm to solve the object model and identify all possible scenarios and their associated probabilities. Since scenario likelihoods are developed directly by the solution algorithm, they need not be computed by statistical inference based on Monte Carlo observations (as required by some discrete event simulation methods). Thus, OBEST is not only much more computationally efficient than these simulation methods, but it also discovers scenarios that have extremely low probabilities as a natural analytical result – scenarios that would likely be missed by a Monte Carlo-based method. This report documents the OBEST methodology, the demonstration software that implements it, and provides example OBEST models for several different application domains, including interactions among failing interdependent infrastructure systems, circuit analysis for fire risk evaluation in nuclear power plants, and aviation safety studies.

*Equilibrium is when all of the fast things have happened
and all of the slow things have not.*

— Richard Feynman

Acknowledgments

No significant research work can be completed without contributions from a number of people, and this project has been no exception. The authors wish to thank a number of people who have provided help for this work. First, we would like to thank the Sandia National Laboratories Laboratory-Directed Research and Development program, Infrastructure Interdependencies research area, for providing the financial sponsorship that made this work possible.

Many other people at Sandia contributed their time, leadership, comments and criticisms to this work, and this project would be much the poorer without their help. We would like to thank Allen Camp for his leadership and encouragement throughout this project. Persons who provided important input very early in the OBEST methodology development process (input that helped refine the methodology in almost every way) are too numerous to mention, but included Roger Breeding, Roger Cox, Sharon Daniel, William Hart, Cindy Phillips, Mark Snell, and the staff of the Surety Assessment Center. John Forester and Dwight Miller deserve special thanks because they not only provided assistance with human reliability analysis topics associated with the OBEST methodology, but also served as reviewers of this final report. Frank Wyant also deserves special thanks for his help and patience with regard to developing the OBEST circuit analysis example described in Section 5.4.

Another unsung member of the project team was Mr. Robert Browitt of Architrave Software in Albuquerque, New Mexico. Robert wrote the OBEST demonstration analysis software described in Chapter 4. Robert did an admirable job of implementing a complex methodology into working software in the face of frequent changes on a limited budget. We are very thankful for his efforts and the software that they produced.

We would also like to thank the NASA Ames Research Lab and the NASA Aviation Safety Program for sponsoring aviation safety work that helped shape and test the OBEST methodology (part of which is reported in Section 5.6 of this report). We would especially like to thank Dr. Mary Connors and Dr. Irving Statler of NASA Ames, as well as Dr. Kevin Corker of San Jose State University for their help, comments and support. We would also like to thank ATAC Corporation of Sunnyvale, California, and especially Mr. Michael Abkin, for their insights into runway incursion phenomenology.

We would also like to thank Ms. Paula Sperling, certified AmPRO racquetball instructor, for her help in developing the racquetball game example found in Section 5.3. We very much appreciate her help in developing the probabilities used in that example as well as her review of its final write-up.

Finally, we would like to thank Mr. David Craner, a pilot with SB Air in Albuquerque, New Mexico. Dave generously gave us assistance in characterizing pilot actions under a variety of conditions. His contributions helped us dramatically in construction of the OBEST aviation safety model that is presented in this report. We would also like to thank the management of SB Air for encouraging and supporting Dave's participation in this project.

This Page Intentionally Blank

Contents

Nomenclature	18
1. Introduction	19
1.1. Motivation	20
1.2. Overview of the Report	22
2. Survey of Other Methods	25
2.1. Infrastructure Indications and Warnings System	25
2.2. Aspen	26
2.3. Traditional Network Analysis Methods	27
2.4. Enumerative Risk Assessment Methods	28
2.5. Probabilistic Risk Assessment Methods	30
2.6. Markov Models	32
2.7. Simulation Methods	33
2.8. Dynamic Risk Assessment Methods	35
2.9. Discrete Event Simulation Methods	36
2.10. Object-Oriented Risk Assessment Methods	38
2.11. Summary	39
3. The OBEST Methodology	43
3.1. Background	43
3.2. Methodology Description	45
3.2.1. Common Elements with Previous Methods	45
3.2.2. The OBEST Action Language	50
3.2.3. OBEST Methodology Summary	55
3.3. Solving an OBEST Model	57
3.3.1. IRM Branching Methodology	60
3.3.2. DRM Evaluation and Branching Methodology	62
3.3.3. Truncation and Binning	67
3.4. Compatibility With Parallel Processing	69
3.5. Comparisons With Other Methodologies	69

3.6.	Potential OBEST Applications.....	72
3.7.	OBEST Limitations and Possible Extensions	74
3.8.	Summary	76
4.	OBEST Software Implementation	79
4.1.	Software Overview.....	79
4.2.	Entering a Basic Object Model	79
4.3.	Entering Immediate and Delayed Response Models.....	89
4.4.	Evaluating an OBEST Model.....	97
4.5.	Summary	105
5.	Example Problems	107
5.1.	Electric Power Supply for a Police Station	108
5.1.1.	Problem Description.....	109
5.1.2.	Object Description.....	109
5.1.3.	Graphical Model Solution	113
5.2.	Fuel Tank Supply for a Gas Burner	114
5.2.1.	System Structure and Interactions.....	114
5.2.2.	The Valve Object	116
5.2.3.	The Fuel System Object.....	118
5.3.	A Singles Racquetball Game.....	120
5.3.1.	System Structure and Interactions.....	121
5.3.2.	Player A Object	122
5.3.3.	Player B Object	126
5.3.4.	The Game Object	129
5.3.5.	Modeling Application and Alternatives	132
5.4.	Circuit Analysis for Nuclear Power Plant Cable Fires.....	132
5.4.1.	System Structure and Interactions.....	135
5.4.2.	Load Object.....	137
5.4.3.	Conductor Object	138
5.4.4.	Switch Object.....	139
5.4.5.	Power Supply Object.....	140
5.4.6.	Partial Circuit Object.....	141
5.4.7.	Circuit Completer Object	142
5.4.8.	Circuit Analyzer Object	144
5.4.9.	Other Objects.....	147
5.5.	Basic Infrastructure Problem.....	148
5.5.1.	Problem Description and System Model.....	148
5.5.2.	Individual Object Models.....	154
5.5.3.	Summary and Insights.....	163
5.6.	Aircraft Spacing During Runway Crossing Operations	164

5.6.1. System Structure and Interactions.....	165
5.6.2. Airport/Airspace Object.....	167
5.6.3. Landing Aircraft Object.....	169
5.6.4. Taxi Aircraft Object.....	172
5.6.5. Tower Controller Object.....	175
5.6.6. Environment Object.....	175
5.6.7. Model Applications.....	175
5.7. Summary.....	176
6. Conclusions and Future Directions.....	177
6.1. Conclusions.....	177
6.2. Recommendations for Future Work.....	179
6.3. Summary.....	181
7. References.....	183

Appendix

A. Example Details for Electric Power Supply for a Police Station.....	189
A.1 System Structure and Interactions.....	189
A.2 Electric Power Bus Object.....	190
A.3 Telephone Switch Building Object.....	191
A.4 Police Station Power Object.....	193
B. Example Details for Fuel Tank Supply for a Gas Burner.....	197
B.1 System Structure and Interactions.....	197
B.2 Valve Object.....	198
B.3 Fuel System Object.....	199
C. Example Details for A Singles Racquetball Game.....	203
C.1 System Structure and Interactions.....	203
C.2 Player A Object.....	204
C.3 Player B Object.....	208
C.4 The Game Object.....	212

D. Example Details for Circuit Analysis for Nuclear Power Plant Cable Fires	215
D.1 System Structure and Interactions.....	215
D.2 Load Object.....	215
D.3 Conductor Object	218
D.4 Switch Object.....	219
D.5 Power Supply Object.....	221
D.6 Partial Circuit Object.....	222
D.7 Circuit Completer Object	223
D.8 Circuit Analyzer Object	225
E. Example Details for Basic Infrastructure Problem	237
E.1 System Structure and Interactions.....	237
E.2 The Vital Household Functions Objects	241
E.3 The Consumable Supplies Objects.....	245
E.4 The Commercial Power and Commercial Gas Objects.....	254
F. Example Details for Aircraft Spacing for Runway Operations.....	259
F.1 System Structure and Interactions.....	259
F.2 Airport/Airspace Object	261
F.3 Landing Aircraft Object	264
F.4 Taxi Aircraft Object	267

List of Figures

Figure 3-1. A System Structure Diagram.....	46
Figure 3-2. An Interaction Diagram.....	47
Figure 3-3. A State Transition Diagram.....	47
Figure 3-4. An Action Data Flow Diagram.....	48
Figure 4-1. OBEST Software Main Screen.....	80
Figure 4-2. Object Definition Screen.....	81
Figure 4-3. Entering Object States.....	82
Figure 4-4. Entering a State Transition Table.....	84
Figure 4-5. Defining a Model Event.....	86
Figure 4-6. Summary of OBEST Model Attributes.....	87
Figure 4-7. Entering Attribute Definitions.....	87
Figure 4-8. Entering Values for Attributes.....	88
Figure 4-9. Entering an Immediate Response Model.....	91
Figure 4-10. Entering a Delayed Response Model.....	92
Figure 4-11. Entering Logic Cases for Immediate and Delayed Response Models.....	93
Figure 4-12. Entering Script Definitions for Immediate and Delayed Response Models.....	95
Figure 4-13. Setting Alternate Initial Conditions for the Model.....	98
Figure 4-14. Setting Events to Initialize the OBEST Model.....	98
Figure 4-15. Setting Path Termination Conditions.....	99
Figure 4-16. Binning Model Results Based on Attribute Values.....	101
Figure 4-17. OBEST Software Display During Model Analysis.....	103
Figure 4-18. OBEST Software Debugging Screen.....	104
Figure 4-19. Displaying the Results of an OBEST Analysis.....	105
Figure 5-1. System structure diagram for police station power model.....	110
Figure 5-2. Interaction diagram for police station power model.....	110
Figure 5-3. State transition diagram for the electric power bus.....	111
Figure 5-4. State transition diagram for telephone switch building.....	112

Figure 5-5. State transition diagram for police station.....	113
Figure 5-6. Graphical Solution of the police station – electric power problem.	114
Figure 5-7. Fuel tank supply for a gas burner.	115
Figure 5-8. System structure diagram for gas burner fuel supply model.....	115
Figure 5-9. Interaction diagram for gas burner fuel supply model.	115
Figure 5-10. State transition diagram for the valve.....	116
Figure 5-11. State transition diagram for fuel system.....	118
Figure 5-12. System structure diagram for model of a singles racquetball game.....	121
Figure 5-13. Interaction diagram for model of singles racquetball game.	121
Figure 5-14. State transition diagram for Player A object.	123
Figure 5-15. State transition diagram for Player B object.....	127
Figure 5-16. State transition diagram for Game Object.	130
Figure 5-17. Example solenoid-operated valve control circuit.	134
Figure 5-18. System structure diagram for circuit analysis.	135
Figure 5-19. Interaction diagram for circuit analysis model.....	137
Figure 5-20. State transition diagram for the Load Object.	138
Figure 5-21. State transition diagram for the Conductor Object.....	138
Figure 5-22. State transition diagram for the Switch Object.....	139
Figure 5-23. State transition diagram for the Power Supply Object.	140
Figure 5-24. State transition diagram for the Partial Circuit Object.	141
Figure 5-25. State transition diagram for the Circuit Completer Object.....	143
Figure 5-26. State transition diagram for the Circuit Analyzer Object.....	145
Figure 5-27. System structure diagram for the “Unprepared” situation.	149
Figure 5-28. System structure diagram for the “With Generator” situation.	150
Figure 5-29. System structure diagram for the “No Electricity” situation.....	150
Figure 5-30. System structure diagram for the combined model.....	151
Figure 5-31. Interactions diagram for the combined model.....	151
Figure 5-32. State transition diagram for the Light Object.	155
Figure 5-33. State transition diagram for the Heat Object.	155
Figure 5-34. State transition diagram for the Cooking Object.....	155
Figure 5-35. State transition diagram for the Water Object.....	156
Figure 5-36. State transition diagram for the Electricity Object.	156
Figure 5-37. State transition diagram for the Camp Fuel Object.	157

Figure 5-38. State transition diagram for the Bottles Object.	158
Figure 5-39. State transition diagram for the Batteries Object.	158
Figure 5-40. State transition diagram for the Candles Object.....	159
Figure 5-41. State transition diagram for the Kerosene Object.....	159
Figure 5-42. State transition diagram for the Firewood Object.	160
Figure 5-43. State transition diagram for the Matches Object.	160
Figure 5-44. State transition diagram for the Generator Object.....	161
Figure 5-45. State transition diagram for the Commercial Power Object.....	162
Figure 5-46. State transition diagram for the Commercial Gas Object.....	162
Figure 5-47. System structure diagram for model of runway operations.....	165
Figure 5-48. Interaction diagram for model of runway operations.	166
Figure 5-49. Physical layout for the Airport/Airspace Object.	168
Figure 5-50. State transition diagram for the Airport/Airspace Object.....	168
Figure 5-51. State transition diagram for the Landing Aircraft Object.....	170
Figure 5-52. State transition diagram for the Taxi Aircraft Object.....	172

Appendix

Figure A-1. System structure diagram for police station power model.	189
Figure A-2. Interaction diagram for police station power model.....	190
Figure A-3. State transition diagram for the electric power bus.	191
Figure A-4. State transition diagram for telephone switch building.	192
Figure A-5. State transition diagram for police station.....	194
Figure B-1. Fuel tank supply for a gas burner.....	197
Figure B-2. System structure diagram for gas burner fuel supply model.	197
Figure B-3. Interaction diagram for gas burner fuel supply model.....	198
Figure B-4. State transition diagram for the valve.....	199
Figure B-5. State transition diagram for fuel system.	200
Figure C-1. System structure diagram for model of a singles racquetball match.	203
Figure C-2. Interaction diagram for model of a singles racquetball match.....	204
Figure C-3. State transition diagram for Player A object.....	205
Figure C-4. State transition diagram for Player B object.....	209
Figure C-5. State transition diagram for Game Object.	213

Figure D-1. Example solenoid-operated valve control circuit.....	215
Figure D-2. System structure diagram for the circuit analysis model.....	216
Figure D-3. Interaction diagram for the circuit analysis model.....	217
Figure D-4. State transition diagram for the Load Object.....	218
Figure D-5. State transition diagram for the Conductor Object.....	219
Figure D-6. State transition diagram for the Switch Object.....	220
Figure D-7. State transition diagram for the Power Supply Object.....	221
Figure D-8. State transition diagram for the Partial Circuit Object.....	222
Figure D-9. State transition diagram for the Circuit Completer Object.....	223
Figure D-10. State transition diagram for the Circuit Analyzer Object.....	225
Figure E-1. Components and interactions in the “Unprepared” situation.....	237
Figure E-2. Components and interactions in the “With Generator” situation.....	238
Figure E-3. Components and interactions for the “No Electricity” situation.....	238
Figure E-4. Components and interactions for the combined model.....	239
Figure E-5. Functional interaction diagram for the combined model.....	239
Figure E-6. States, attributes, and response models for the Light Object.....	241
Figure E-7. States, attributes, and response models for the Heat Object.....	242
Figure E-8. States, attributes, and response models for the Cooking Object.....	243
Figure E-9. States, attributes, and response models for the Water Object.....	244
Figure E-10. States, attributes, and response models for the Camp Fuel Object.....	246
Figure E-11. States, attributes, and response models for the Bottles Object.....	247
Figure E-12. States, attributes, and response models for the Batteries Object.....	248
Figure E-13. States, attributes, and response models for the Candles Object.....	249
Figure E-14. States, attributes, and response models for the Kerosene Object.....	250
Figure E-15. States, attributes, and response models for the Firewood Object.....	251
Figure E-16. States, attributes, and response models for the Matches Object.....	252
Figure E-17. States, attributes, and response models for the Generator Object.....	253
Figure E-18. States, attributes, and response models for the Electricity Object.....	254
Figure E-19. States transition diagram for the Commercial Power Object.....	255
Figure E-20. State transition diagram for the Commercial Gas object.....	256
Figure F-1. System structure diagram model of runway operations.....	259
Figure F-2. Interaction diagram for the model of runway operations.....	260
Figure F-3. Physical layout for the Airport/Airspace Object.....	263

Figure F-4. State transition diagram for the Airport/Airspace Object.	263
Figure F-5. State transition diagram for the Landing Aircraft Object.	264
Figure F-6. State transition diagram for the Taxi Aircraft Object.	268

List of Tables

Table 3-1. A Deterministic Truth Table.....	49
Table 3-2. A Probabilistic Truth Table	50
Table 3-3. Notional Representation of an Immediate Response Model	51
Table 3-4. Notional Representation of a Delayed Response Model	52
Table 3-5. Equivalence of Models Using the “If – Then – Else” Decision Process.	54
Table 3-6 Improper Model Simplification Using the “If – Then – Else” Decision Process.	55
Table 5-1. Functional interactions for police station power model.....	110
Table 5-2. Functional interactions for gas burner fuel supply model.	116
Table 5-3. Probabilistic schedule for turning on the valve.	117
Table 5-4. Probabilistic schedule for turning off the valve.....	117
Table 5-5. Delay times for changing tanks.	119
Table 5-6. Functional interactions for model of singles racquetball game.	122
Table 5-7. Player A’s serve profile.	124
Table 5-8. Player A’s serve return profile conditional on Player B’s serve.....	124
Table 5-9. Player B’s serve profile.....	127
Table 5-10. Player B’s serve return profile conditional on Player A’s serve.....	128
Table 5-11. Functional interactions for circuit analysis model.	136
Table 5-12. Probabilities for adverse conditions of completion.	144
Table 5-13. Objects in the combined structure diagram.	152
Table 5-14. Functional interactions within the combined model.....	153
Table 5-15. Functional interactions for model of runway operations.	166
Table 5-16. Collision potential for combinations of runway and taxiway locations.	169
Table 5-17. Probabilistic timing for aircraft movement in State:(Moving).	171
Table 5-18. Probabilistic timing for aircraft movement in State:(Taxi).	174

Appendix

Table A-1. Functional blocks for police station power.	189
---	-----

Table A-2. Functional interactions for police station power model.....	190
Table B-1. Functional blocks for gas burner fuel supply.....	198
Table B-2. Functional interactions for gas burner fuel supply model.....	198
Table C-1. Functional blocks for singles racquetball game.....	203
Table C-2. Functional interactions for model of singles racquetball match.	204
Table D-1. Functional blocks for the circuit analysis model.	216
Table D-2. Functional interactions for the circuit analysis model.	218
Table E-1. Functional interactions for the combined model.....	240
Table F-1. Functional blocks for runway operations.	260
Table F-2. Functional interactions for runway operations.	261

Nomenclature

DETAM	Dynamic event tree analysis method
DRM	Delayed response model
DYLAM	Dynamic logical analytical methodology
ETA	Event tree analysis
FMEA	Failure modes and effects analysis
FMECA	Failure modes, effects and criticality analysis
FTA	Fault tree analysis
HAZOP	Hazards and operability analysis
IRM	Immediate response model
LDRD	The Laboratory-Directed Research and Development program at Sandia National Laboratories
LHS	Latin hypercube sampling
OBEST	Object-based event scenario tree
OO	Object-oriented
OPRRA	Object-oriented process for risk and reliability analysis
PRA	Probabilistic risk assessment

OBEST: The Object-Based Event Scenario Tree Methodology

1. Introduction

For more than 30 years, the science of risk assessment has been used to identify potential shortcomings and safety concerns for a wide variety of engineered systems. The objective of a risk assessment is generally to:

- Identify the scenarios and mechanisms that can push an engineered system away from its designed or desired behavior into a region of undesired or unintended consequences,
- To categorize those scenarios or mechanisms based on their likelihood of occurrence, and
- To provide mitigation strategies that will reduce either the likelihood or consequence of scenarios and mechanisms that are found to be most important.

Thus, at its core, risk assessment can be thought of as seeking the answers to three fundamental questions:¹

- What can happen to my system?
- How likely is it to happen?
- How does it affect my system, its operators, its surroundings, and the larger environment?

Typically, a risk assessment focuses on the “bad” things, i.e., “What are all of the ‘bad’ things that can happen to my system?” and “What are all of possible negative effects of those scenarios?” To answer these questions effectively, one must thoroughly and systematically explore what may be a very large scenario space in order to identify those (hopefully few) scenarios that may lead to undesired consequences.

Over the years, deductive and inductive risk assessment methods have been applied to a wide variety of systems, including missiles, nuclear power plants, underground waste repositories, aircraft, manufacturing processes, information systems, and even consumer goods. Many have also sought to apply these same methods to the assurance of telecommunications, electric power, and other infrastructures. The results of these studies have generally been less insightful than might otherwise have been expected because of the complex interconnections, interactions, and interdependencies both within and among the various infrastructure systems. The U.S. Department of Energy, among others, has organized major research programs aimed at overcoming some of these obstacles.

Realizing that infrastructure protection is an issue with major national security implications, Sandia National Laboratories has allocated significant internal research and development funding toward developing risk assessment methods and risk mitigation technologies for infrastructure applications. In October of 1988, Sandia began a new two-year laboratory-directed research and development (LDRD) project with the goal of developing improved methods for risk and reliability analysis for infrastructure-related problems – particularly for network-based infrastructures. This project was also asked to consider methods that might aid in the understanding of infrastructure interdependencies. The final result of this project was the major methodological advance that is documented in this report.

1.1. Motivation

Risk assessment is fundamentally concerned with identifying and preventing “bad” things from happening to or through engineered systems. It provides a framework to help the analyst identify scenarios that will lead to undesired consequences, the likelihood of each scenario, and magnitude of undesired consequences that would be expected were the scenario to actually occur. Over the years, a wide variety of methods have been developed to support this process, and one can generally divide these methods into two broad categories: deductive methods and inductive methods.

A deductive method starts by identifying a particular undesired consequence and then asking, “What are the immediate, necessary and sufficient conditions that will cause this consequence to occur?” This causal relationship is expressed in terms of a graphical logic diagram that illustrates the combinations of conditions required to cause the consequence. One can then ask the same question about each of these contributory conditions, and expand the logic diagram accordingly. This recursive process continues until all contributory conditions are resolved into their most basic causes. The most common deductive risk assessment technique is fault tree analysis. Once the fault tree² (graphical logic diagram) is completed, it is solved to find the minimal combinations of the events that are both necessary and sufficient to cause the undesired consequence. The graphical diagram illustrates the individual steps in the deductive reasoning process, and allows others to understand not only the results (*how* and *why* things fail), but also the method by which those results were obtained (why *these* elements contribute to system failure). Deductive risk assessment methods have proven difficult to apply to network-based infrastructure analysis problems because complex network topologies can make it difficult to exhaustively identify *all* of the immediate, necessary and conditions that contribute to a particular fault condition (i.e., a fault can traverse many different pathways before manifesting itself as a consequence). In addition, it is difficult to deductively and exhaustively predict the types of cascading failures (the “domino effect”) to which many networks have been demonstrated susceptible.

An inductive risk assessment methodology begins not by identifying a consequence, but by identifying an initial configuration for the engineered system.¹ The risk analyst then systematically examines the space of scenarios that are credibly reachable based on the stated initial configuration. A variety of methods are used to probe this scenario space, including:

- Enumerative methods – methods such as failure modes and effects analysis (FMEA) and hazards and operability analysis (HAZOP) systematically consider the scenarios that result

when individual components or process parameters individually assume abnormal or failed conditions.

- Event sequence methods – methods such as event trees and influence diagrams can be used to develop exhaustive sets of event sequences, but require extensive assumptions to be made with respect to the list of credible events and the order in which they can occur.
- Simulation-based methods – given the existence of the tool for simulating the performance of the engineered system (often a computer software tool), one can drive that tool to model a variety of scenarios and boundary conditions, and in this way, explore the scenario space.^{*}

All of these methods have been used over time as risk assessment tools for network-based infrastructures. However, the results of these assessments have generally been less than satisfying. Enumerative methods, because they consider failures or process abnormalities in a one-at-a-time fashion, are unable to consider the effects of potentially rare but important combinations of failures.[†] Event sequence methods have proven useful in some applications, but an inability to determine *a priori* the order in which events would occur has reduced their usefulness.[‡] Simulation methods are regularly employed using both deterministic and probabilistic scenario selection. However, the computationally intensive nature of the simulation tools has often meant that the number of scenarios to be explored must be severely limited, thus reducing the completeness and, hence, the usefulness of the risk assessment.

The project team began its work by considering a wide variety of existing simulation and risk assessment methodologies as they might be applied to infrastructure analysis problems. We considered several different classes of methodologies, including detailed simulation models, traditional risk and reliability analysis models, contingency analysis-based models, simple quasi-quantitative models of infrastructure interdependencies, and “pure” network analysis methodologies. All of these methodologies were found to be wanting in one way or another (these reasons will be discussed in detail in chapter 2). In examining these methods, we came to understand that network-based infrastructures – particularly those associated with electric power distribution and telecommunications – have a number of characteristics that are very different from the engineered systems for which risk assessment techniques have been most successfully used in the past. These include:

^{*} The scenarios and boundary conditions can be specified *a priori* based on observed incidents, an analyst’s intuition, regulatory or licensing requirements, or the results of another risk assessment method. One can also use statistical distributions to define an entire space of potential scenarios and boundary conditions, and use techniques such as Monte Carlo or importance sampling to generate the details of the particular scenarios to be simulated.

[†] In fact, it was by appealing to the rareness of these combinations of failures that several current “best practice” telecommunications architectures (e.g., the common channel signaling network, SONET rings) were developed.

[‡] Event tree models that support variable event ordering are very complex to build and maintain, while simply building multiple models to represent a variety of possible event orderings is very labor-intensive.

- The presence of long periods of quasi-equilibrium behavior interspersed with relatively short periods of rapid system change.
- The importance of *combinations* of low probability events whose importance and consequences are difficult to predict *a priori*.
- The capacity for network-based infrastructures to exhibit congestion-based and cascading failures that are not amenable to discovery using deductive methods.
- The strong dependence of system behavior on event ordering, with the *a priori* identification of event order – as required for event tree analysis – either impossible or infeasible.*

Because of these characteristics (which are not necessarily unique to infrastructure analysis problems), the project team sought a new and different risk assessment methodology that would be more compatible with these types of system constraints. The object-based event scenario tree (OBEST) methodology was the result of that search. OBEST utilizes concepts from traditional event tree analysis, object-oriented analysis (as used in computer science), and discrete simulation to obtain a method that produces probabilistically-weighted scenarios (like the event tree method) yet supports on-the-fly determination of event ordering (like the simulation method), all the while encapsulating model details and behaviors within an object-oriented analysis framework. While the OBEST methodology does not fulfill all of the requirements of infrastructure analysis, it does provide important advances for our risk assessment capabilities and complements the existing methods.

As the project team has exercised the OBEST methodology, it has become clear that OBEST can be applied to a class of problems that is much broader than just infrastructure assessment. The method has already been applied to aviation safety analysis and the assessment of circuit and equipment performance in the presence of electric circuit fires. In each case, the OBEST method has enabled analysts to develop new insights that would not have been possible[†] using previously extant methods.

1.2. Overview of the Report

This report is organized in seven chapters. Chapter 2 contains a broad survey of some of the risk assessment and simulation methods that the project considered when searching for an appropriate infrastructure risk assessment methodology. It describes the advantages and drawbacks of each methodology, its applicability to infrastructure risk assessment problems, and notes selected characteristics of the method that should either be included in or excluded from a new

* Indeed, the order in which many events occur is determined based on “race conditions,” such as whether off-site electric power is restored before onsite batteries are depleted. These conditions have historically been difficult for logic-based risk assessment methods to properly characterize.

[†] Or could have been gained only using labor-intensive methods that would, in practice, be cost-prohibitive.

infrastructure risk assessment methodology. Chapter 3 describes the OBEST methodology, its history, its potential applications and limitations, and compares OBEST with the methods described in Chapter 2. Chapter 4 describes how the OBEST methodology was implemented in a demonstration software package.*

Chapter 5 provides several examples to show how the OBEST methodology can be applied in a variety of technology domains. The example problems range from modeling a racquetball game to consideration of aircraft behavior under icing conditions. Finally, Chapter 6 contains suggestions for future research as well as a summary and conclusions of this study.

* Note that, while the demonstration software package has been successfully used to test the methodology and selected applications, it is not yet production quality and, for this reason, is not available for distribution.

This Page Intentionally Blank

2. Survey of Other Methods

The purpose of this chapter is to provide the reader with background regarding the inductive risk assessment methods that are currently in use. As each method is described, its respective advantages and disadvantages will be noted, as will characteristics of the method that should either be included in or excluded from a new methodology. This will provide motivation for why the OBEST methodology was developed. This “list of recommendations” will be reexamined in Chapter 3 as we describe the OBEST methodology in detail.

2.1. Infrastructure Indications and Warnings System

Sandia National Laboratories has developed an Infrastructure Indications and Warnings System³ for the rapid analysis of threats to critical infrastructures. This Indications and Warnings System is a PC-based quick-look tool to help analysts generate a quick and reasonable assessment of their susceptibility to infrastructure-related disruptions. The methodology embodied in this system is a relatively simple and quasi-quantitative representation of the interdependencies between various infrastructure elements as obtained by an interview-like expert judgement-based process. It constructs a map of interdependencies among the various infrastructure elements, and allows an analyst either to enter “real” circumstances that might affect these interdependencies, or to play “what if” games on those interdependencies. As circumstances are entered into this system, quantitative weighting factors, which are developed from expert judgement interviews, determine how those circumstances might affect other parts of the infrastructure. Depending on the fidelity of the weighting factors and underlying mathematical formulae, the Indications and Warnings System can help the analyst determine whether this set of circumstances is more or less severe than some other set of circumstances. In this way, the analyst may be able to infer whether this situation should be of concern. Since the Indications and Warnings System does not perform a traditional time-based simulation of any scenario, it cannot reasonably be expected to infer a time line of events toward the scenario’s future outcome (nor is it intended to).

A key feature of Sandia’s Indications and Warnings System is that it is fast. As a rapid analysis tool, it allows an analyst to quickly understand the gravity of a current situation and explore a series of contingency plans – likely in real-time. Thus, the Indications and Warnings System can play a role in the area of situation awareness and response planning. In fact, it may be possible to tie a tool such as this to an optimization engine and generate optimized response plans to a complex situation.^{*,4} This speed, however, comes at a price. Since the Indications and Warnings System does not perform time-based simulations, it is unable to predict scenarios that might befall a system as combinations of normal events, random failures, and external events interact with the system. Experience with rapid tools such as the Indications and Warnings

* Genetic optimization engines have been linked to risk assessment models in other disciplines with some success, as noted in Reference 4. Specifically, the risk assessment results have been incorporated into the utility function to allow computation of system configurations and spare parts inventories that are optimized with respect to cost, system availability, minimal mean-time-to-repair, etc.

System shows that one must carefully balance the need for rapid execution and analysis with the desire to develop insights from a problem-specific scenario database. A new modeling methodology should be flexible to allow the analyst to select any level of analysis detail (few high-level results to many detailed scenarios) as appropriate for the problem and situation at hand.

2.2. Aspen

Sandia's second well-known infrastructure analysis methodology is the Aspen^{5, 6} economic simulation tool. In one sense, the Sandia Indications and Warnings System and Aspen methods can be viewed as two extremes on a continuum. Whereas the Indications and Warnings System model is quasi-quantitative and does not employ time-based scenario simulation, the Aspen model is a highly complex direct simulation of an economic system that is dependent on a variety of infrastructures. Aspen is designed to run on a parallel-processing machine, and the simulation of a single scenario can take many hours of computational time even on the fastest machines. Aspen embodies the behavior of the various actors within an economic system as intelligent agents, which are capable of learning appropriate behaviors based on achieving their economic interests. Once these agents are trained, they are allowed to interact with each other over a series of time steps. As each agent simulates its own behavior (making probabilistic decisions as necessary), the collective action of all agents causes the behavior of the economic system to emerge. This emergent behavior, then, embodies the response not only of the infrastructure systems themselves, but also of the rest of the economy to disruptions in those infrastructure systems.

Aspen provides investigators with a powerful approach for understanding the behavior of systems that are so complex that they have heretofore defied simulation using conventional approaches. The ability of Aspen to model the consequences of infrastructure failures not just within the infrastructure itself, but within the larger picture of a national economy makes Aspen a valuable tool for those tasked to plan policies, future system capacities, and possibly even regulatory issues. However, while Aspen provides great insights into the behavior of economic systems in the presence of infrastructure uncertainties, the fact that Aspen takes hours to analyze a single scenario – even on today's fastest computers – makes it infeasible to simulate a large number of scenarios using Aspen. This conflicts with the traditional risk assessment goal of developing a systematic and complete picture of the possible system scenario space. Thus, it is very difficult to use Aspen results as the sole basis for an infrastructure risk analysis.^{*, 7} A new methodology should be able to simulate many scenarios within the time required for a single Aspen simulation. This method should also, however, support the use of results from detailed simulations such as Aspen to inform and correct its model. It should also be able to flag specific

* Aspen can be used in infrastructure risk assessment studies in much the same way that MELCOR is used in reactor risk assessment studies. The results from a limited number of detailed simulation studies provide the analyst with insights into the behavior of the system, which can then be translated into simpler models that must be executed to explore the larger scenario space.

scenarios that cannot be adequately resolved using the faster-running model for analysis using the more detailed simulation tools.

2.3. Traditional Network Analysis Methods

Existing network analysis methods were also investigated as a point of departure for this research project. The project team examined both traditional and nontraditional network analysis models, including those with and without capacity constraints, k -terminal^{8, 9} and all-terminal^{10, 11, 12} connectivity models, as well as models from the field of operations research. The methods range from simpler capacity analysis tools (e.g., “Is there enough generation and transmission capacity in the network under these conditions to meet that postulated load?”) to more complex time-dependent simulation tools that assess specific network characteristics (e.g., voltage and phase stability, call completion probability, dial tone delay) using time steps that can be as small as milliseconds.

Several network analysis methods are in use at Sandia in both production analyses and research activities, and many more are being routinely used by industry. These network connectivity and capacity models form the basis for planning and analysis models within individual infrastructures (e.g., electric power or telecommunications*),^{13, 14} often in the context of “contingency analysis” studies. In a contingency analysis,¹⁵ particular combinations of known, observed, and/or hypothetical fault conditions called “contingencies” are inserted into a network, and a network analysis tool is used to determine whether this compromised network is capable of meeting its objectives. These contingencies can be selected deterministically (e.g., single contingency analysis might systematically examine the effects of all single faults on the system as they are applied one-at-a-time), probabilistically (e.g., through Monte Carlo sampling), or heuristically based on the desires or concerns of the analyst. Contingency analysis techniques represent the state-of-the-art in many infrastructure areas – particularly for electric power grid stability analysis. Optimization techniques can be applied to these types of problems, with the results from network analysis algorithms forming a portion of the utility function being evaluated.

While network analysis methods are widely applied within individual infrastructures, they have not seen significant application *between* infrastructures. This is because common examples of these methods assume that the network being analyzed is capable of carrying *all* of the modeled commodities over the *same* network infrastructure. Indeed, the more typical examples of these methods assume that these different commodities are in fact competing for available capacity within a single network, as would different types of goods for space in a limited supply of rail cars. This situation does not apply well to the interdependencies between infrastructures because the different infrastructure networks are generally incapable of carrying goods or traffic related to another infrastructure. Our project decided to focus on methodologies that would allow analysts to gain insights across infrastructures. Since different types of network models are

* Queuing theory models are also prominent in the telecommunications industry for analysts seeking to understand the performance of a switch or router under heavy loading conditions.

being developed and adapted regularly to solve the problems specific to individual infrastructures, we felt it most appropriate to concentrate on other types of models, while paying particular attention to opportunities to interface with these specialized and well-documented methods where benefits could be realized.

The advantage of a network analysis modeling paradigm over more traditional logic-based risk assessment models (to be discussed in the next few sections) is that a network analysis tool inherently deals with the fact that a highly interconnected network behaves in a manner that is much different from the behavior of the sum of the individual elements of which it is composed. This aggregate behavior can be very difficult to understand – let alone predict – using the reductionist techniques that are most commonly used in traditional risk assessment. Many naïve risk assessment researchers have boldly stated that they could, for example, use a fault tree to model an electric power network, only to find out that the complex interplay between network elements is not easily captured by the independent events necessary for a fault tree structure.

The principal disadvantage of many network analysis methodologies, as applied to risk assessment, is that they merely simulate the conditions within the network – they describe what happens within the network without providing insights as to *why* it happens. The analyst specifies the system initial and/or boundary conditions, and the analysis tool tells us what happens. It is only through a large number of scenario simulations that the analyst starts to identify which elements are most important to the network’s health and which conditions might be precursors to major system failure events.* A new infrastructure risk assessment methodology should provide a mechanism to interface with these specialized network assessment models so that the reductionist or scenario development risk assessment engine can take advantage of the insights from these detailed network models without having to recreate their capabilities (either as part of the automated model solution package or, through the analyst, as part of the model itself).

2.4. Enumerative Risk Assessment Methods

The failure modes and effects analysis^{1, 16} (FMEA) technique, along with its close cousins, failure modes, effects and criticality analysis (FMECA) and HAZOP,^{1, 17} are generally the first systematic risk and reliability analysis techniques applied to any system – infrastructure or not. The purpose of a FMEA is to examine individual components and assess the effect of their failure on the system in which they are used and on other systems and subsystems. FMEA is a qualitative method that is typically documented in a tabular format. To accomplish a FMEA, the analyst goes through the components of a system one by one, and for each component considers

* This discussion applies primarily to network analysis models that are tuned to simulate a particular type of network (electric power, for instance). It should be noted that other classes of network analysis methods are more generalized and provide insights into optimal network configuration, capacity constraints, etc. However, since these more generalized methods are often more distant abstractions of the actual physical system being modeled, their results need to be validated using more detailed and specialized network performance models in order to gain credibility.

every known failure mode individually. The analyst writes a description of the failure mode itself, the method which would detect that failure in the operating system, the effect of the failure on the system or subsystem, and the expected response of operators or automatic controls to the situation. Elucidating comments are also included in order to allow others to understand the full scope and gravity of the situation caused by the component's failure.

The FMEA documentation is often extended by including extra information in the analysis table. Typical extensions include a quantitative or qualitative estimate of the likelihood of each assessed component's failure, a qualitative categorization of the criticality of the effects caused by the component failure, and the possible actions to reduce the failure rate or effects. This is often called a FMECA. In such an analysis, one can rank the results in terms of either the likelihood or the criticality of the component failure scenarios. A more recent ranking method combines the likelihood and criticality descriptors to obtain a "risk descriptor" that is low for improbable, low-criticality events, and increases as either or both of the descriptors become large. These risk descriptors can then be used as a basis for determining whether remedial action should be taken to reduce the likelihood or criticality of the scenario.

A HAZOP study is related to a FMEA or FMECA in that it assesses predefined scenarios to determine their probable causes, consequences, and possible remediation actions. It also typically includes qualitative assessments of criticality, likelihood, and risk similar to those described above. However, while FMEA and FMECA studies focus on the effects of individual component failures, the HAZOP method focuses on qualitative deviations of key system operating parameters from their nominal, normal, or design values. The fundamental philosophy here is that normal operations are inherently safe, and deviations are the source of unrecognized problems. The scenarios that can lead to these deviations are arrived at through a combination of systematic consideration, deductive logic (to obtain probable causes for the parameter deviations), and imaginative thinking. The objective is to find the "weak link" in the system, and to provide a basis for developing procedural or engineering controls to reduce any risks so identified.

The principal advantage of these techniques resides in their simplicity. Virtually anyone can apply these techniques – and often with little training. The results from these analysis techniques are intuitive and easy to explain to non-technical or management personnel. However, because they consider failures and/or parameter deviations in a one-at-a-time manner, these techniques are unlikely to discover those obscure scenarios that involve multiple failures or compounding human errors. History tells us that, while design engineers often focus on preventing consequences from catastrophic failures that involve one (or even a few) very rare but credible events, major consequences are more frequently caused by an unanticipated combination of more likely (but seemingly more benign) failures – often complicated by one or more human errors. Thus, while these enumerative risk assessment methods can provide valuable insights into system operations, they are generally believed to be insufficient for ensuring the safety of high-consequence systems.

Another important disadvantage of these techniques is related to their reliance on a "bottom-up" problem-solving method. By this we mean that the sources of risk are identified at the *beginning* of the analysis, instead of being inferred by a systematic deductive "top-down" analysis such as would occur in a fault tree analysis. If the risk analyst does not think of a particular scenario,

and the mechanics of the analysis or the “best practice” checklist does not drive them to identify it, then that scenario is likely to remain unanalyzed because the scenarios are the starting point of the analysis, *not* its result. A new infrastructure risk assessment methodology should ideally infer the sources of risk as a result of the analysis. This, however, is usually infeasible for inductive assessment methods. A more attainable goal is for the new assessment method to automatically and systematically generate a thorough list of potential scenarios (and their likelihoods), and, to the degree possible, to include the effects of all potential failure modes in those scenarios as is done in the enumerative risk assessment methods.

2.5. Probabilistic Risk Assessment Methods

Other types of models considered for infrastructure risk assessment included traditional probabilistic risk assessment (PRA) models. If the objective is to examine the universe of possible scenarios that can result from a particular set of initial conditions within the interconnected infrastructure, then it is important to examine the set of inductive probabilistic risk assessment modeling techniques.^{*} While there are many such techniques in existence, the two that are most closely applicable to this situation are event tree analysis^{18, 19, 20} and influence diagrams.^{21, 22}

Event tree analysis (ETA) seeks to represent an undesired occurrence as a sequence of events. Event trees are similar in form to decision trees, and are used to represent the spectrum of possible outcomes given a particular initial condition. The method is inductive in that it begins with a set of initial conditions and uses inductive logic to infer its results. Each path through the event tree is constructed by selecting a unique outcome for each event within the event tree model. Thus, the path physically represents a unique sequence of events so that outcome O_{1P} occurs for event 1, *and* outcome O_{2P} occurs for event 2, *and* outcome O_{3P} occurs for event 3, and so forth.[†] If the event tree model is properly constructed, the set of all paths through the model represents the complete set of possible outcomes that can occur as a result of the given initial condition (but typically only the outcomes relevant to the analyst’s needs).

The events within an event tree may include the status of physical systems, operator actions, the activities of automated control systems, and random (stochastic) events both internal and external to the system. The events may represent simple yes-or-no questions (“binary events” such as, “Does the operator turn the system on?”), or they may involve multiple possible outcomes (“multi-branch events” such as, “Which of the five displays does the operator check first?”). The events may or may not be independent of one another. If the events are not independent, then the dependencies between them are explicitly included within the logical structure of the tree.

^{*} The discussion in this section focuses exclusively on inductive probabilistic risk assessment methods. Deductive methods such as fault tree analysis were also examined but found unsuitable as described in Section 1.1 on page 20.

[†] The subscript “P” indicates that these outcomes are characteristic of this specific event tree path, and will vary from path to path in the event tree solution.

The results of an event tree analysis are initially qualitative in that each path defines a scenario in terms of the outcomes for individual events. If, however, one assigns conditional probabilities to the various outcomes such that $P(O_{2p})$ is actually the conditional probability that O_{2p} occurs *given* that O_{1p} has already occurred, then one can also obtain quantitative results consisting of the scenario (path) definition and its probability of occurrence.

An influence diagram is a probabilistic network that consists of nodes and arcs. The nodes can represent system states, decisions, or chance or deterministic occurrences, while the arcs represent the conditional dependencies among these occurrences. The nodes ultimately influence a “value node” that quantifies the consequences for each possible combination of occurrences and system states. Conditional probabilities can be applied within the nodes to represent the probability that a particular event happens *given* particular conditions in the other nodes to which it is connected (i.e., states, decisions, or events that *influence* this node). Thus an influence diagram consists of four distinct parts: the nodes, the influences upon the nodes (the dependencies among the nodes, as represented by the arcs), the conditional dependencies within each node upon other nodes in the model, and the conditional probabilities themselves.

The influence diagram method is conceptually similar to the event tree, decision tree, and fault tree methods described earlier. It can be applied as both an inductive and a deductive modeling tool in that one can begin either with the value node (the objective, as is done with fault tree analysis) or with a suitable initial condition (as is done with event tree and decision tree analysis). One could even begin with some of each and work both inductively and deductively as necessary until the model is complete. In addition, the method is not limited to simple binary events as is fault tree analysis (FTA). This flexibility makes the influence diagram an important risk analysis tool.

Historically, the most important disadvantage to the use of influence diagrams has been that the traditional solution method does not show or even generate the detailed set of scenarios or paths possible in the model. The ability to examine these paths in detail is a primary advantage of the FTA and ETA methods. However, it has been noted that there always exists a (non-unique) transformation from an influence diagram model into an event tree model. When applied, this allows an analyst to use the advantages of influence diagram construction to produce the more desirable results associated with ETA.

The ETA solution methodology and results have several highly desirable characteristics. First, the results of an ETA are a complete set of scenarios that can occur as a result of a particular set of initial conditions.* Second, the solution methodology is recursion – something that is both well understood and readily adaptable to parallel processing. Third, the solution methodology

* For very large event trees (those with thousands or even millions of possible pathways), the complete set of scenarios is not generated. Scenarios can be “truncated” from the analysis if they are found to be too unlikely, do not result in consequences, or have some other characteristic that renders them beyond the scope of the risk assessment study. Similar scenarios are also summarized or “binned” with one another to reduce the number of paths to a number that can be examined by a mere mortal.

needs to discover each scenario only once – and develops the scenario’s likelihood at the same time (this is much different from a Monte Carlo-based discrete event simulation, which will be discussed in Section 2.8 on page 35). Finally, the methodologies explicitly embody time and order conditions (something FTA cannot easily do). These characteristics should ideally be present in a new method to assess risk for infrastructure systems.

Upon examination, it was determined that both of these methods suffer from the same fundamental problem when applied to infrastructure analysis problems, albeit in different ways. Event trees and influence diagrams are both, by definition, acyclic graphs. Since cycles are prohibited, it can be difficult to represent the behavior of systems that embody feedback loops in these types of models. However, feedback loops are common within the interconnected infrastructure system, and the behavior of these feedback loops is critical to the understanding of the possible scenarios that can result. The presence of feedback loops manifests itself in event trees as non-determinism in the ordering of the event tree events. Thus, to model such a feedback system using event trees, one must construct several event trees – one for each possible ordering of the events. These feedback loops manifest themselves in influence diagrams as a violation of the acyclic connectivity requirement. The ability to handle at least some types of feedback loops should be an important part of any new infrastructure risk assessment methodology.

2.6. Markov Models

Markov models¹⁶ are directed graphs that capture the concepts of system states and probabilistic transitions between states. To build a Markov model, an analyst examines every relevant configuration of a system – both functional and nonfunctional configurations – and defines them to be *states* of the system. The analyst then defines the probability of transition from each state to every other state (as a function of time and other factors) to complete the model. State transitions that are precluded for physical reasons are assigned a transition probability of zero. In their simplest and most common forms, Markov models use constant state transition probabilities, although Markov models with time-varying state transition probabilities have also been used.²³

Markov models provide a natural, direct representation, through the use of cycles, of systems whose components are repairable and systems where component failures have interactions. They also readily embody the feedback characteristics found in many infrastructure problems. Recall that fault trees and event trees are acyclic graphs and hence do not readily accommodate these system characteristics. The two basic forms of Markov models are chains and processes. A Markov chain uses matrix multiplication in discrete time to obtain state transition probabilities. A Markov process uses a set of differential equations over continuous time. Relative to the other techniques discussed, Markov processes require a more sophisticated understanding of mathematics for their solution. In fact, most Markov models of real systems suffer from “state explosion” and hence are difficult to solve, requiring simulation. Complete path or scenario information is not a natural output of a Markov model.

An interesting application of Markov modeling is found in the continuous event tree methodology.^{24, 25} In this method, the branching operations within an event tree model are viewed as state transitions within the framework of a Markov model. This allows the analyst to

determine the population of each state (and, hence, of each branch within the event tree model) as a function of time. The method has been extended to a semi-Markov process to allow for the state and branch transition probabilities to vary as a function of the length of time the system has spent in that state.²⁶

The main advantages of Markov models lie in their ability to identify the probability that a system will exist in a particular state as a function of time – even in the presence of repairable failures and inherent system feedback. Their main disadvantages from an infrastructure risk assessment perspective lie in the difficulty with which path or scenario information is extracted from their results, the state explosion that occurs when large systems are analyzed, and the difficulty of the mathematical methods required to solve them. A new infrastructure risk assessment method should avoid state explosion (path explosion is difficult enough to deal with), and should be solvable with straightforward mathematical methods.

2.7. Simulation Methods

The logical risk assessment methodologies described in the previous few sections require a human analyst to inductively or deductively interpret the physical characteristics of a system and embody those characteristics in a logical description (graphical or tabular). Logical models are often capable of rapidly examining a broad cross-section of potentially important system behaviors. A complementary approach to logic models involves the direct use of more traditional simulation models (often software-based).^{*} The simplest application of simulation technology to risk assessment involves an *a priori* definition of “important” scenarios, which are then evaluated using an appropriate simulation tool to assess the system’s performance and/or consequences. The scenarios may be selected based on incidents previously observed for similar systems, questions raised by a FMEA or other logic model, or the judgment of domain experts, or a regulatory authority may simply prescribe them. Regardless of their source, the objective of such analyses is to characterize the behavior of the system under a variety of (presumably) important normal and abnormal operational conditions.

Simulation is also frequently used in conjunction with logic models to determine the consequences associated with particular failure scenarios. In this application, evaluation of a logic model (e.g., fault trees or event trees) produces a large number of scenarios that may potentially lead to undesired consequences. Some or all of these scenarios may be simulated to determine the degree of consequences produced (lives lost, environmental damage, etc.). Because the logic models may produce far more scenarios than one can afford to simulate, analysts commonly group scenarios such that a number of “similar” scenarios are represented by a single surrogate scenario, the consequences of which are assumed to apply reasonably well to the entire group of similar scenarios. This methodology has been used to assess the radiological consequences of nuclear reactor accidents for more than two decades.²⁷

^{*} The use of simulation models for risk assessment can be thought of as a generalization of some of the network analysis methodologies described in Section 2.3 on page 27.

More complex applications of simulation technology for risk assessment involve the automated selection of the scenarios to be simulated. If, for example, a system is intended to operate safely for all conditions within a particular range of temperature and humidity conditions, one might use a mathematical design of experiments technique to ensure that the entire range of temperature and humidity combinations was appropriately represented in the scenarios to be simulated. Or, one might accomplish the same goal through probabilistic sampling of temperature and humidity conditions using Monte Carlo, Latin hypercube, or importance sampling techniques.^{28, 29} Other methods such as dynamic event trees or continuous event trees²⁵ work toward a similar objective using analogies to the ETA methodology that is popular as a PRA tool (see Section 2.8). When using these techniques, the importance of particular parameters to the overall set of simulation results can be inferred through straightforward statistical regression analysis techniques. Probabilistic techniques are often less efficient when the number of parameters to be considered is small, but can have significant advantages when a simulation is based on a large number of independent variables.

To this point, we have discussed only techniques that define the list of scenarios to be simulated prior to the first simulation being conducted. These techniques do not use feedback from previous simulations to select additional scenarios for consideration. However, a number of feedback techniques are available to answer specific risk-related questions. For example, adaptive importance sampling can be used to generate additional scenarios in regions of the parameter space that produce “interesting” results (e.g., high-consequence regions of the parameter space).^{30, 31} Quasi-analytic reliability methods such as the mean-value method and first-order and second-order reliability methods can be used to answer questions such as, “For what range of input parameters can I be 95 percent certain that I will not produce an unacceptable consequence?”³² Genetic algorithms can be used to zero in on particular parameter combinations that will most closely produce a particular set of results (e.g., the most economical operating conditions, the highest consequences, or the lowest disturbance to the general public).⁴ The use of feedback methods has, until recently, been relatively limited because of the computationally intensive nature of many simulation packages (one could not afford to run the number of simulations that might be required by such a method). Recent improvements in both computer performance and numerical algorithms, however, have made these methods far more broadly accessible.

The advantages of using simulation models in risk assessment are that, if properly validated, they can (theoretically) provide the most accurate depiction of how a scenario will progress of any of the risk assessment models described in this chapter. The disadvantage of simulation approaches lies in the need to use other methods to determine the scenarios that are to be analyzed using the simulation model. Depending on the scenario selection method, these can be highly inefficient and can fail to cover the entire credible scenario space. In addition, if the scenario selection method does not support likelihood determination (the *ad hoc* methods, for example), then this key element of risk must be determined using other methods (often it is estimated by the judgement of an expert). Finally, the computational effort required to evaluate large numbers of scenarios using detailed simulation tools may be well beyond the computational capabilities of many organizations especially if the simulation tools are computationally intensive). A new infrastructure risk assessment methodology should automatically interface with simulation tools so that the risk assessment engine can take advantage of the simulation results without having to recreate their capabilities (either within the automated model solution package or, through the

analyst, as part of the logic model itself). Ideally, it should also seek to minimize the number of simulations that must be performed and draw the maximum amount of information from each of the required simulation runs.

2.8. Dynamic Risk Assessment Methods

In the 1980s and early 1990s, analysts sought to overcome the limitations inherent in both the inductive PRA methods (ETA, Markov modeling) and the simulation methods described in the previous few sections. Briefly stated, ETA and Markov models provided a mechanism for dealing with the time-dependent aspects of scenarios, but the time dependence they provided was often imposed through the filter of a human analyst, and might be at odds with the “real” time history of the scenario as predicted by a simulation model. However, methods for varying the inputs to simulation models ranged from *ad hoc* to Monte Carlo, but lacked the scenario branching, random failures, and orderly scenario evolution found in ETA. Analysts also wanted to examine time-dependent human responses to scenario development using computational models. The Dynamic Logical Analysis Method (DYLAM) and Dynamic Event Tree Analysis Methodology (DETAM) were developed to help meet these needs.

A concise description of the DYLAM method was presented by Siu in Reference 33.

The general procedure used to employ DYLAM is conceptually direct. First, a physical model of the system is constructed. If needed, a model for the operating crew is also constructed. The physical model predicts the response of system process variables to changes in component status (e.g., hardware failures). It is constructed by linking physical models for system components, where the component models predict the changes in process variables due to changes in component states. A model for a pump, for example, may consist of a set of equations for the pump head, each equation applying to a different pump state. ...

The next step is to define the undesired system states. These are defined in terms of process variable levels, rather than hardware states, and are used to determine when a particular accident scenario should be terminated.

Finally, the system model is used to simulate all possible accident sequences as follows. Starting at $t = 0$ and some user-defined initial state, the system model is used to determine the change in process variables (if any) in the next Δt At the end of the first time interval $(0, \Delta t)$, all possible combinations of component states, i.e., all possible system hardware states, are identified and their likelihood calculated. These states are then used to provide boundary conditions for the next round of process variable updating; the generation of an event sequence continues in this manner until an absorbing state is reached.³³

DETAM is very similar to DYLAM in most respects. Acosta and Siu note that “the primary difference is that DYLAM does not emphasize the stochastic nature of operator behavior as strongly [as DETAM].”³⁴

Applications of both DYLAM and DETAM have found the need to limit the expansion of the event tree model in order to maintain the computational feasibility of the problem.* This was done by limiting the method to consideration of a single failure at each time step (as opposed to “all possible combinations of component states” as described above) and by truncating low probability scenarios from further development. Additional computational feasibility issues were found to arise when the characteristic time scale of the problem is short compared to the overall scenario duration.³³ Since infrastructure problems are often characterized by short periods of intense activity (short characteristic time scale) followed by long periods of quasi-equilibrium activity (long scenario duration), this methodology is of questionable value for modeling infrastructure failure scenarios.

Another important limitation of these methodologies is found in the prescriptive nature of their branch point timing requirements. DYLAM considers component state changes only at fixed times (exact multiples of the analyst-specified Δt). DETAM allows the analyst to specify particular times at which component state changes might occur (in addition to the fixed multiples of Δt used in DYLAM). However, both DETAM and DYLAM allow branching events to occur *only* at times that are prescribed *prior* to the start of the analysis and not at times that might be inferred from the characteristics of the simulation model. Neither method allows the simulation to request a scenario branching operation. This restriction has a number of consequences. For example, it is impossible to accurately implement the random failure of a component to start on demand using these methods unless that demand, by coincidence, occurs at an exact multiple of Δt or at a time selected by the analyst *prior* to the start of the simulation.

The principal advantages and disadvantages of these methods derive from the fact that they are inextricably tied to dynamic simulation models. Since these methods make use of true simulation models to predict system parameter values, the accuracy of their results is limited only by the fidelity of the simulation models upon which they are built. At the same time, many of these simulation models are computationally intensive, so using DYLAM or DETAM to specify many executions of such a model can rapidly become impractical. The time scale issues described previously are also particularly problematic for the infrastructure failure events that are the subject of this study. An ideal computational methodology should support the modeling of systems that operate simultaneously on a variety of time scales. It should allow for multiple simultaneous failure events at times that are either prescribed by the analyst or inferred by the simulation. Finally, an ideal computational methodology should allow the system to evolve at whatever size time step is appropriate without imposing unnatural restrictions on the value of Δt .

2.9. Discrete Event Simulation Methods

Another class of methods that can be used for infrastructure analysis is sometimes called probabilistic discrete event simulation. One non-infrastructure instantiation of this method is in

* Siu notes that these methodologies are limited to treating accidents where the number of possible scenarios is relatively small.³³

the LASEP-T software³⁵ that was used to simulate launch phase accidents for the Cassini space probe. A similar simulation methodology is used in the SimMod-Pro aviation safety analysis software.^{36, 37, 38} We will use LASEP-T as an example to describe the discrete event simulation method. A LASEP-T model can be thought of as a series of probabilistic decisions, each of which is described by one or more statistical distributions. As a scenario reaches one of the probabilistic decision points, LASEP-T draws a random number to represent a Monte Carlo “decision process.” It uses this random number to decide which of the discrete events represented in the decision point will actually occur. A decision point may represent, for example, discrete events such as whether a particular cladding element is breached, or the type of material upon which a particular element lands. Continuous variables are also selected using Monte Carlo methods. Examples of such variables include the time at which the booster rocket self-destructs, the size of a cladding breach, the number of milligrams of radioactive material released, and the particle size distribution for that release. A scenario is constructed by using the results from many of these probabilistic decisions to determine an exact time-dependent sequence of events that are presumed to occur. The Monte Carlo-based scenario development process is then repeated over many trials to obtain a probabilistic representation of the characteristics of the overall scenario space being investigated.

The LASEP-T model relies on a Monte Carlo process to make all of its decisions. The Monte Carlo nature of these decisions means that each of the many scenarios discovered is, under the assumptions of Monte Carlo simulation, equally likely. Aggregating the results of a large number of such equally likely scenarios provides a probabilistic view of the set of possible scenarios given a particular initiating event. This result is very close to what is sought for infrastructure analyses.

The disadvantages of a Monte Carlo-based simulation method, however, include the requirement that one simulate an extremely large number of scenarios if one is to identify potentially rare, high-consequence scenarios. For example, to identify all scenarios whose frequencies are approximately one in 1000 per launch, one may have to run hundreds of thousands or millions of scenarios with the software. In addition, if one scenario is dramatically more likely than all others, the Monte Carlo nature of this simulation requires it to be identified over and over again in order to preserve the statistical significance of the results. Also, the Monte Carlo nature of this software makes it difficult to apply traditional uncertainty analysis techniques to this process.²⁹ While one could apply biasing schemes to the Monte Carlo simulation, the fundamental fact remains that this methodology is a computationally inefficient way to identify rare scenarios. A new infrastructure risk assessment method should avoid the pitfalls of a pure Monte Carlo scenario definition approach in order to more efficiently identify those rare scenarios that may result in high consequences to the infrastructure system. Ideally, a new method should compute scenario probabilities explicitly and identify each scenario only once in order to maintain reasonable computational efficiency.

2.10. Object-Oriented Risk Assessment Methods

An LDRD project that was completed in October of 1998 provided a new paradigm that synthesized aspects of probabilistic risk analysis and object-oriented (OO) analysis methodologies.^{*, 39} The basic idea behind OO modeling^{40, 41, 42} is to encapsulate the behavior of a component or subsystem entirely within the confines of a self-contained “object.” Objects contain *attributes* that describe its changeable characteristics, as well as *states* and behavioral models that describe how the object reacts to internal and external stimuli. Objects communicate with one another by exchanging *messages*, which can represent the transfer of information, materials, or energy. A model of the entire system is composed by combining and connecting the object models for the individual components or subsystems of which it is comprised (i.e., enabling the individual objects models to communicate with one another in a way that faithfully replicates their interactions in the real world system).

In an object-based risk assessment model,^{†, 39} one builds an object model to represent the behavior of the system to be analyzed (including normal and abnormal modes of operation, deterministic, and probabilistic behavior). This system object model can be either decomposed into object models for individual subsystems or components, or it can be composed from generic object modules that have been developed as standardized representations for common components. This ability to compose an object model from a library of generic components represents a significant labor saving advance in risk assessment methodology.

Once the object model has been constructed, the analyst can apply particular boundary conditions to the model and then query it to extract a wide variety of risk models that have similar characteristics to those generated using traditional inductive and deductive risk assessment methodologies. Since the object model embodies the *behavior*, or causality, of the system (rather than abstractions of *parts* of that behavior, as in traditional logic-based risk assessment models), one can extract risk models to answer many different questions under a broad array of conditions from a single object model. System boundary conditions can be changed quickly without invalidating the object model – the analyst simply re-extracts the appropriate risk or surety models based on the revised boundary conditions. This object-oriented method forms an inviting framework for many types of surety models because it enables the analyst to build a *single* model of the system, instead of building a separate risk or surety model for each question that is to be answered.

While these methodological characteristics are inviting, there are several problems to be overcome before these methods can be applied to the types of infrastructure problems envisioned

* The development and application of object models is described more fully in Section 3.1, beginning on page 43. Object-oriented analysis methods were originally developed in the computer science arena to aid in the specification and development of complex software.

† The object-oriented analysis method developed in Reference 39 is a derivative of the Shlaer-Mellor methodology, although the UML methodology was also investigated.

under the current project. The chief obstacle is that the methodology development activities under the former project did not advance to the point where the element of time was adequately embodied. In other words, the methods were derived initially for static and quasi-static systems,* and significant extensions would be required to adapt the methodology to systems and scenarios that exhibit strong time dependence, such as the infrastructure systems envisioned under this project. In addition, the methodology has yet to be fully embodied in software because it is still the subject of ongoing research.

A number of insights from this methodology should be considered in a new infrastructure risk assessment methodology. First, the ability of an analyst to model the behavior of a system and then query it in a variety of ways enables truly meaningful model reuse to occur. The savings in analyst and quality assurance time can be dramatic. Second, the ability to compose a useful model quickly from a library of generic components provides a second opportunity for major labor savings. And finally, while the ability to extract many different *types* of risk models from a single object model was an important goal of the previous work, its resulting attempt to be “all things to all people” made it less than ideal for many types of problems. A new methodology should focus on doing a particular class of problems well – better than any other method can – and consider extensions that broaden its appeal, instead of doing a mediocre job of solving all classes of problems.

2.11. Summary

In this chapter we have briefly described a variety of types of network analysis and risk assessment techniques that have been suggested for application to infrastructure risk assessment problems. The techniques have included the Sandia National Laboratories Indications and Warnings System, Aspen, traditional network analysis methods, enumerative risk assessment methods (FMEA, HAZOP, etc.), inductive probabilistic risk assessment methods (event tree, influence diagrams, etc.), Markov models, traditional simulation-based methods, discrete event simulation methods, and object-oriented risk assessment methods. We have noted the advantages and disadvantages of each class of methods, and attempted to identify the role that each might play in the science of infrastructure risk assessment.

We have also noted insights from each of these methods that provide guidance for the development of a new infrastructure risk assessment method. Based on those insights, the characteristics of such a new method should ideally be as follows:

- A new methodology should be flexible to allow the analyst to select different levels of analysis detail (few high-level results to many detailed scenarios) as appropriate for the problem and situation at hand.

* Or, more precisely, for systems that can be embodied in terms of state machines that are time-invariant.

- A new methodology should be able to simulate many scenarios within the time it might take for a highly detailed simulation model to perform a single simulation.
- A new methodology should support the use of results from detailed simulation models to inform and correct its model.
- A new methodology should be able to flag specific scenarios that cannot be adequately resolved using its faster-running model for analysis using the more detailed simulation tools.
- A new methodology should provide for a direct interface with specialized network assessment models in order to take advantage of their insights without having to recreate their capabilities (either as part of the automated model solution package or, through the analyst, as part of the model itself). It should provide interfaces to other types of simulation models as well.
- A new methodology should seek to minimize the number of simulation and/or network analysis calls that must be performed. It should also draw the maximum amount of information from each of the required simulation runs.
- A new methodology should ideally infer the sources of risk as a result of the analysis, rather than requiring the analyst to list them at the beginning of the analysis process. This, however, is usually infeasible for inductive assessment methods. A more attainable goal for the new assessment method is to automatically and systematically generate a thorough list of potential scenarios (and their likelihoods) and, to the degree possible, include the effects of all potential failure modes in those scenarios as is done in the enumerative risk assessment methods.
- A new methodology should, to the extent possible, mimic some of the desirable characteristics of event tree analysis. First, its results should be a complete set of scenarios that can occur as a result of a particular set of initial conditions. Second, it should use a simple solution methodology – one that is both well understood and readily adaptable to parallel processing. Third, it should discover each scenario only once – and develop the scenario's likelihood at the same time. Finally, it should explicitly model time and order conditions.
- An important part of any new methodology is the ability to properly model at least some types of feedback loops.
- A new methodology should avoid state explosion (path explosion is difficult enough to deal with) and should be solvable with straightforward mathematical methods.
- A new methodology should support the modeling of systems that operate simultaneously on a variety of time scales. It should allow for multiple simultaneous failure events at times that are either prescribed by the analyst or inferred by the simulation. It should also allow the system to evolve at whatever size time step is appropriate without imposing unnatural restrictions on the value of Δt .

- A new methodology should avoid the pitfalls of a pure Monte Carlo scenario definition approach in order to identify rare scenarios efficiently. Ideally, a new method should compute scenario probabilities explicitly and identify each scenario only once in order to maintain reasonable computational efficiency.
- A new methodology should, to the extent possible, seek to embody the desirable characteristics noted for the object-oriented risk assessment methodology. It should embody the behavior of the system directly rather than requiring an analyst to abstract out only the “risk-important” behaviors. It should enable the risk modeling of many different scenarios by varying the boundary conditions for a single object model rather than requiring the construction of separate models for the different scenarios. It should also enable the construction of a library of “generic” objects that can be used as basic model building blocks for future analyses (i.e., model fragment reuse). Finally, it should be able to incorporate both deterministic and probabilistic behavior.
- A new methodology should enable the analyst to model the *behavior* of a system and then query it in a variety of ways to enable truly meaningful reuse of models. It should also enable the analyst to compose a useful model quickly from a library of generic components. Finally, the new methodology should focus on doing a particular class of problems well – better than any other method can – and consider extensions that broaden its appeal, instead of trying to be all things to all people by doing a mediocre job modeling an extremely broad class of applications from the start.

OBEST was developed with many of these requirements in mind. This methodology is described in the following chapter. Where appropriate, that chapter compares the characteristics of OBEST to the above list of desired model characteristics, and identifies the tradeoffs that were made in developing that methodology.

This Page Intentionally Blank

3. The OBEST Methodology

The preceding chapter described how we identified what we believe to be a gap in the analysis technologies available to solve infrastructure risk assessment problems. We sought a tool or methodology that would enable one to systematically examine a wide variety of infrastructure scenarios from probabilistic basis without resorting to computationally inefficient Monte Carlo-based discrete event simulations. As we were unable to find such a method, we began development of a new analysis methodology. Our goals for that method were to enable an analyst to probabilistically explore the “universe” of possible scenarios that might arise as a result of a set of initial conditions in much the same way as a traditional event tree analysis. However, unlike a traditional event tree analysis, we believed that an infrastructure risk assessment method must support the development of event scenarios with variable event ordering. We also sought a method that would be based directly on a *behavioral* model of the system (such as an object-based model) so that the analyst could perform a wide variety of analyses without having to manually construct and validate many different models. We also sought a method that would determine the likelihood of each possible scenario based on a single computation. We wanted to avoid Monte Carlo-based event simulation methods because their use inherently means that 1) it is extremely difficult to detect low frequency failure combinations (very large numbers of scenarios must be generated and assessed), and 2) if a single scenario dominates the scenario space, it will be selected and analyzed many times during the Monte Carlo analysis, thus wasting significant computational effort.

The result of our methodology development effort is the object-based event scenario tree methodology, or OBEST. OBEST utilizes concepts from traditional event tree analysis, object-oriented analysis (as used in computer science), and discrete simulation to obtain a method that produces probabilistically-weighted scenarios (like the event tree method) yet supports on-the-fly determination of event ordering (like the simulation method), all the while encapsulating model details and behaviors within an object-oriented analysis framework. While the OBEST methodology does not fulfill all of the requirements of infrastructure analysis, it does incorporate many of the desirable features described above and provides important capability advances for our risk assessment capabilities.

3.1. Background

As its name implies, the fundamental basis of the OBEST methodology is an object-oriented (OO) approach to model representation. In an object-oriented modeling paradigm, objects are model constructs used to represent real world entities that can “communicate” with one another. This communication is considered to consist of messages exchanged between objects. Messages can represent the transfer of information, materials, or energy. When an object receives a message, it responds by altering its internal state (i.e. its underlying behavior changes in a fundamental way) and/or by generating outbound messages to communicate its conditions to the other objects in the model. The way in which the object responds to messages depends on its internal processes and on its internal state. The basic idea, then, is to encapsulate the behavior of a component or subsystem entirely within the confines of a self-contained object. A model of the entire system is composed by combining and connecting the object models for the individual

components or subsystems of which it is comprised (i.e., enabling the individual objects models to communicate with one another in a way that faithfully replicates their interactions in the real world system).

The object-oriented analysis methodology from which OBEST is most closely derived is a modified version of the Shlaer-Mellor methodology,⁴⁰ as revised and extended for the object-oriented process for risk and reliability analysis (OPRRA) described in Reference 39. Under the Shlaer-Mellor methodology, the analyst begins to model a system by identifying the objects that populate the system and documenting the relationships among these objects using a *system structure diagram* and an *entity relationship diagram* (sometimes also called a *system interaction diagram*). These diagrams establish the structure of the overall system. In an infrastructure problem, for example, an object may be a telephone central office, a water pumping or storage station, or an electric power bus or transmission line. For each object identified, the analyst determines whether the behavior of the object changes or is invariant over time. If its behavior changes, the analyst documents these changes using a *state transition diagram*. This diagram identifies the *events* that trigger changes in behavior and labels the behavior that the object exhibits as it enters its new state (the analyst defines a new state for each fundamentally different behavior that can be displayed by the object). Within each state, the analyst describes the object's behavior using an *action data flow diagram*. The non-behavioral characteristics of the object that can change over time are represented in *attributes*, which, for Shlaer and Mellor, can be continuous, discrete, or Boolean variables. The Shlaer-Mellor method also supports concepts of *object communication models*, *synchronous access models*, *domains*, *bridges*, and other types of *messages*, but these concepts are beyond what is required to describe the OBEST modeling methodology. Thus, in this sense, OBEST can be regarded as a simplification of this well-established computer science analysis methodology.

When developing the OPRRA method,³⁹ the authors sought to use object-oriented methods to create a generalized risk assessment framework. In that study, the authors came to realize that Shlaer and Mellor's concept of action data flow diagrams was not powerful enough to support a robust inductive and deductive risk assessment methodology. The greatest problems were found in the inability to trace deductively backwards through the action data flow diagram and conclusively state that, "since this result has occurred, that particular set of causes must have been present." In addition, since the Shlaer-Mellor methodology was created to support the development of software packages, it was not designed to support the nondeterministic and probabilistic behaviors that are so important to risk assessment. Therefore, when developing the OPRRA methodology, the authors replaced Shlaer and Mellor's action data flow diagram concept with probabilistic truth tables. This enabled the analyst to enter nondeterministic and probabilistic behaviors into the object model in support of inductive risk assessment modeling while at the same time enabling backwards tracing through the model in support of deductive risk assessment modeling. This did have the effect of limiting the new OPRRA methodology to discrete values for attributes (something not required in the Shlaer-Mellor methodology), but since many risk assessment methods are already inherently discrete, this was not viewed as a serious limitation.

In the current study, we have focused on further modifications to the behavioral modeling aspect of the object-oriented analysis methodology. These modifications were motivated by an observation about the characteristics associated with failures in infrastructure related systems.

We noted that infrastructure failure scenarios do not involve continuous degradation of infrastructure system performance. Rather, a system will exist in a quasi-equilibrium condition for some extended period of time (possibly minutes, hours, or days), and then exhibit some very rapid and dramatic change in behavior. Furthermore, these rapid changes frequently cause rapid changes in other parts of the system – both within this infrastructure and, potentially, in other infrastructures.* Within a very short period of time (possibly seconds or milliseconds), the various interconnected components and subsystems respond to these changing conditions and come to a new quasi-equilibrium condition. This new equilibrium lasts for an extended period of time until another dramatic change occurs,† and the process repeats itself. This observed pattern of behavior became the basis for a replacement to the behavioral modeling paradigms of Shlaer and Mellor (action data flow diagrams) and OPRRA (probabilistic truth tables). This new behavioral model is foundational to the OBEST modeling methodology.

3.2. Methodology Description

3.2.1. Common Elements with Previous Methods

To document the system’s functionality,‡ the methodology described in this document uses four types of views: a system structure diagram, an interaction diagram, a state transition diagram, and a data flow diagram. The system structure diagram, presents the “chunks” of functionality found in the system and specifies how these chunks (components, subsystems, etc.) relate to one another (Figure 3-1). In Figure 3-1, a *system structure diagram*, each of the blocks (M through P) represents some portion of the functionality delivered by System A (which could itself be a component in some higher level system). For the purposes of this discussion, it is assumed that these functions are embodied in individual components within System A (this is often true for “real-world” systems). When taken together, these blocks embody the entire functionality of System A. Lines between any two blocks indicate that those blocks interact with one another. As needed, each block in a diagram can be further subdivided. This process of hierarchical decomposition of the system is carried as far as is needed for the analyst to adequately document the functional structure of a system.

* Note, for example, that the failure of an electric power bus can cause power failure in a telephone central office and force a dramatic change of behavior to the telephone infrastructure as the central office attempts to switch over first to emergency generators, then to batteries, and finally, may experience a power failure that causes loss of call processing capabilities.

† For example, the emergency generator runs out of fuel or the batteries become fully discharged.

‡ Major portions of the OBEST methodology are essentially identical to the OPRRA methodology described in Reference 39. This section quotes heavily from that document.

Figure 3-1. A System Structure Diagram

A second view used to describe a system's functionality is the *interaction diagram*, as shown in Figure 3-2. This diagram's function is to capture the dynamics of the system at "black box" levels. Using the blocks produced in the system structure diagrams, interaction diagrams show the relevant "flows" that occur among various functional blocks in the model. These flows can represent materials (i.e., tangible things), energy, or information. Depending on the nature of the analysis to be done, the labels on the flows (e.g., A, B on the upper left flow) will identify either attributes of the flows ("messages") that occur between two blocks, or will represent objects that flow from block to block.* For example, if component N outputs a chemical and component O consumes it, then flow A might specify attributes of the chemical that is flowing (e.g., the quantity of chemical, the chemical's temperature and composition). The result of this flow (as will be discussed below) could be to change the internal attributes of components N and O (e.g., the amount of the chemical that each block stored). The concept here that is important from a modeling point of view is that a flow occurred that resulted in changes in the attributes associated with the components involved in the flow.

* An analyst may model a flow as an object if the behavior of what is flowing is important. For example, component N might represent the process that produces a sterile container. Flow B represents the containers that flow between components N to O (O might document the process by which the containers are filled). Here, the analyst might model the containers as objects with (e.g., capacity, contents) and behaviors (e.g., "accept contents" or "dispense contents"). A flow then becomes a framework for describing how an object moves from one relationship (i.e., with the process that creates it) to another (i.e., with the process in which it is filled).

Figure 3-2. An Interaction Diagram

In an OO modeling methodology, an analyst often develops a self-contained model entity called an *object* to represent the behavior embodied in each functional block or component. An object can exhibit a variety of different behaviors, where a behavior is defined as a unique set of responses to stimuli. Each different type of gross behavior is represented in a *state*, and the ways in which these gross behaviors change are called state transitions. State transitions are caused by *events*, and are documented in a *state transition diagram*. In this type of diagram, the “states” might be represented by rounded rectangles and the “events” represented by italicized words, as shown in Figure 3-3. Events may be externally generated (e.g., the arrival of a flow at a block) or internally generated (e.g., the block reaching a given state, set of attribute conditions, or even a given point in time being reached). The “transitions” specify the state to which an object moves if it receives the required event while in that state. Note that any state can be associated with multiple transitions.

Figure 3-3. A State Transition Diagram

At this point, one should differentiate between the characteristics of the object that describe the object’s behavior from those that describe changeable characteristics of the object itself. Those that describe changeable characteristics of the object are embodied in *attributes*. For Shlaer and

Mellor, attributes can be thought of as variables within the object model that exist regardless of the object's state.* These variables can be continuous, discrete, or Boolean representations of a measurable or calculable characteristic (e.g., temperature = 98.6° F, the traffic light is green, or "the light switch is off" is false). In the OPRRA method, attributes were restricted to discrete and Boolean values. This restriction is continued in the initial implementation of the OBEST methodology.

To this point, we have noted that a state represents a situation where the object embodies a particular set of behavior, and an attribute represents the object's measurable and calculable characteristics. Next, the analyst must describe the details of the object's behavior while it is in each state – how the object responds to stimuli and the ways its attributes change. For Shlaer and Mellor, this is accomplished using an *action data flow diagram*. As shown in Figure 3-4, the diagram consists of four elements. The circles represent processes that transform flows. The lines represent the interconnections between processes and carry the various flows, which are identified by labels. The flows themselves are described as collections of attributes. The horizontal lines represent a "store" which contains the attributes that document the current state of the functional block (e.g., a functional block representing a chemical process might contain a store with attributes that specify the current composition, temperature, quantity, etc., of the chemicals in the process).

Figure 3-4. An Action Data Flow Diagram

In order to enable the use of probabilistic branching and deductive logic, the OPRRA method required that the analyst implement the transformations within Shlaer and Mellor's data flow

* Attributes may also be used to record an object's state since an object's attributes can be examined by other objects while its state cannot. However, even here, the attribute itself exists regardless of the object's state – it just assumes a new value to reflect the change in the object's state.

diagram as truth tables, as shown in Table 3-1. The function of a truth table is to specify the value a given attribute assumes when the input attributes on which the attribute depends assume particular combinations of values. It is a straightforward task to develop this specification for attributes that are limited to discrete values. When the attributes assume a continuous range of values, the task is a little trickier. To address this issue, the research team borrowed the notion of “landmark values” from the qualitative physics community.⁴³ In this approach to modeling attribute values, the goal is to identify key points in the attribute’s range of values at which behavior somehow changes. For example, if the attribute under consideration is the amount of fluid in a storage tank, then it is reasonable to assume that the tank might exhibit three different behaviors corresponding to the tank’s being empty, to the tank having some amount of liquid in it, and to the tank’s being full. Given this, two landmark values would be assigned to the tank (i.e., “tank empty” and “tank full”).

Table 3-1. A Deterministic Truth Table

Number of items in stock — Number of items requested	Number of items produced	“Request Rejection”
+	No. of items requested	F
0	No. of items requested	F
—	0	T

In most cases, a truth table will have a single output column that corresponds to a single attribute on one of the output flows from a process and some number of input columns (corresponding to some or all of the attributes on the input flows to a process). In some cases (e.g., when the process involves a “test”), attribute flows are combined into a single column, as shown in Table 3-1. Also, as shown in this figure, multiple tables may be combined for the sake of compactness if the logic of the tables being combined is sufficiently simple. Note also that, while not shown in this example, a truth table can embody probabilistic behavior. To extend this example, one might envision a situation where an item is in particularly high demand, but a “first come, first served” distribution method is considered unfair (say, for a new life-saving cancer drug). If the distribution method were changed, say, to *randomly* reject two-thirds of all requests and fill only the other one-third using a lottery-like distribution system, one might obtain a truth table like the one shown in Table 3-2.

Table 3-2. A Probabilistic Truth Table

Number of items in stock — Number of items requested	Probability	Number of items produced	“Request Rejection”
+	0.667	0	T
	0.333	No. of items requested	F
0	0.667	0	T
	0.333	No. of items requested	F
—	1.000	0	T

3.2.2. The OBEST Action Language

The preceding brief description of the Shlaer-Mellor and OPRRA methods provides a point of departure for an exposition of the OBEST methodology. The principal areas of difference between OBEST and these other methods are related to the details of the action language for describing object behavior within a particular state. The OBEST action language, which is described in the following paragraphs, replaces Shlaer and Mellor’s action data flow diagrams and OPRRA’s probabilistic truth tables (although it has obvious parallels to the latter, as the reader will note shortly). Recall that, in the OBEST methodology, each object is capable of both immediate and delayed reactions to both internal and external stimuli, and each type of reaction can either be deterministic or probabilistic. In OBEST, these descriptions of object behavior are called the *immediate response model* (IRM) and the *delayed response model* (DRM). Mathematically, these models are identical except that the DRM incorporates a time delay before its ultimate actions are realized. Both the IRM and the DRM can be thought of as probabilistic truth tables similar to those used in the OPRRA method and shown in Table 3-2. And, just as that probabilistic truth table can be thought of, in a “generic” computer programming jargon, as a big “if – then – else if – else” or “select case” construct, the IRM and DRM represent similar ideas and functions. Thus, the structure for an IRM might look like the generic example found in Table 3-3. In this example, each “set of logical conditions” is a logical combination of attribute values from this and/or other objects within the system model. Each “script” is a series of actions that will be taken by the object, and may include the generation of object model events (to cause this or another object to change states), changing the values of its own attributes,*

* Note that, to simplify modeling considerations, each object is only allowed to modify the values of its *own* attributes, and cannot directly modify the values of attributes in other objects. The IRM and DRM models for those objects must be “aware” of the objects around them and change their attribute values accordingly.

and/or generation of special “system events.”* Since scripts can cause an object to change its attribute values, they are specific to the object under investigation. A set of logical conditions, on the other hand, can be used by any number of objects within the model as appropriate because it looks only at object attributes, which are publicly available to all objects within the analysis. Scripts and logical conditions can be reused (simply re-referenced) throughout the object model as appropriate.

Table 3-3. Notional Representation of an Immediate Response Model

<p>If (<i>set of logical conditions #1 is satisfied</i>) Then Probability = xxx[†] : Execute Script A Probability = yyy : Execute Script B Probability = zzz : Execute Script C</p> <p>Else If (<i>set of logical conditions #2 is satisfied</i>) Then Probability = aaa : Execute Script B Probability = bbb : Execute Script D</p> <p>Else If (<i>set of logical conditions #3 is satisfied</i>) Then (<i>Probability = 1.000</i>) : Execute Script A</p> <p>Else (<i>Default</i>) Probability = ccc : Execute Script C Probability = ddd : Execute Script D Probability = eee : Execute Script E</p>
--

A generic example of DRM can be seen in Table 3-4. Note that the principal difference between the IRM structure and the DRM structure is that every script in the DRM is associated with a particular delay. Thus, a line in the DRM model can be thought of as saying, “I, the object, can remain in equilibrium for *X* hours under *this set* of conditions with this *probability*. After that time has elapsed, I will take the actions described in *script*.” Thus, according to the DRM in Table 3-4, if the object model is configured such that “Set of logical conditions #5” is satisfied,

* System events relate to the management of the object model, and may include such events as starting, stopping or resetting delay clocks, or terminating or truncating the evaluation of a scenario.

[†] While the probabilities in this example are scalar, they can also be observations sampled from a distribution using Monte Carlo or Latin hypercube sampling. The OBEST software, however, does not currently support such sampling.

the model says that the object is always able to remain in this configuration for X hours, and when that time has elapsed, will always take the actions described in script A .^{*} Similarly, if the system configuration satisfies set of logical conditions #1, then the object is always able to remain in this configuration for Q hours, and when that time has elapsed, it may take the actions described in script G (with a probability of aaa), or it may use script D (with a probability of bbb). The “Else” situation states that, under these conditions, the object will always take the actions described in script H , but we are unsure whether that will occur after X hours, Y hours, or Z hours. Those delay times will occur with probabilities ccc , ddd , and eee , respectively. Finally, the fully general situation, where probabilities, delays and scripts are all different, would occur if set of logical conditions #4 is satisfied. This allows the analyst full flexibility to express whatever combination of delays and actions might be required to describe the behavior of the system being modeled.

Table 3-4. Notional Representation of a Delayed Response Model

<p>If (<i>set of logical conditions #4 is satisfied</i>) Then Probability = xxx : Delay X hr., then Execute Script A Probability = yyy : Delay Y hr., then Execute Script B Probability = zzz : Delay Z hr., then Execute Script C</p> <p>Else If (<i>set of logical conditions #5 is satisfied</i>) Then (<i>Probability = 1.000</i>) : Delay X hr., then Execute Script A</p> <p>Else If (<i>set of logical conditions #1 is satisfied</i>) Then Probability = aaa : Delay Q hr., then Execute Script G Probability = bbb : Delay Q hr., then Execute Script D</p> <p>Else (<i>Default</i>) Probability = ccc : Delay X hr., then Execute Script H Probability = ddd : Delay Y hr., then Execute Script H Probability = eee : Delay Z hr., then Execute Script H</p>

Note that the notional DRM depicted in Table 3-4 shares some of the scripts, probabilities, and logical conditions used in the notional IRM described in Table 3-3. This sharing is permitted (but not required) between models within the same object – even, if appropriate, between IRMs and DRMs in different states. The sets of logical conditions can also be shared between objects.

^{*} The situation where the system configuration changes before X hours have elapsed such that the required conditions required by logical set of conditions #5 are no longer satisfied will be discussed later in this chapter.

Thus, the behavioral models incorporated in the OBEST model can be represented as a series of logical conditions, each of which is associated with one or more sets of actions (scripts). If a logical condition is associated with more than one potential set of actions, the alternatives must each be given a probability of occurrence. The only difference between an IRM and a DRM is that each potential set of actions in a DRM is delayed by an analyst-specified “equilibrium time” before it occurs.

The “if – then – else if” decision style described in the notional example IRM and DRM has been implemented in the demonstration software that has been developed to implement the OBEST methodology. This decision style mimics that used in the EVNTRE⁴⁴ and SETAC⁴⁵ logical event tree analysis codes that have been developed by Sandia previously. While this decision style may seem straightforward, those who have attempted to use it have often become confused about some of its implications. Note that this decision style is implemented here in the same way it is used in most major computer programming languages. One feature of this implementation is that the computer looks at the sets of logical conditions *in the order specified by within the program* and it *only* executes the instructions associated with the *first* set of conditions that evaluate as “True.” This means that, even if the current set of conditions satisfies more than one set of logical conditions, the computer will only execute the instructions for the first satisfied set that it encounters. It *ignores* all other sets of instructions – even though their logical conditions may evaluate to True. The implications of this fact are demonstrated by the example shown in Table 3-5. The equivalence of these two models demonstrates a powerful fact about the “if-then-else” decision style: the analyst can use this equivalence to simplify the sets of logical conditions used in later parts of the model. While the example demonstrates this in a trivial problem, it should be easy to imagine how, for extremely complex sets of logical conditions, this can provide a dramatic opportunity for model simplification. At the same time, this type of model simplification does present a danger: applying the simplified conditions in the wrong order causes the model to obtain the wrong answer, as demonstrated by the example found in Table 3-6. Therefore, the analyst must be very careful to ensure that the intended logical instructions are actually the ones embedded in the model when using the “if-then-else” decision style to simplify logical models.

One of the many goals stated for a new infrastructure analysis methodology was the ability to incorporate failure modes into a probabilistic risk model. While it may not be obvious, the IRM and DRM provide the mechanisms for this to occur. Consider an example where a telephone central office loses its commercial power supply. It might immediately attempt to start its onsite backup power generator, and, should that fail, it will rely on batteries that will last for a time that is specified by a statistical distribution (i.e., the batteries may fail before they reach their expected discharge time). The demand-related failure of the backup generator can be modeled as a probabilistic transition within the IRM. Here, the analyst provides logical conditions that detect when the situation demands the backup generator start. Associated with those logical conditions are two probabilistic branches – one that executes a script for the situation where the generator starts and a second that executes a script for the situation where the generator fails to start. Similarly, the distribution of failure times for the batteries can represent a discretization of the battery failure time statistical distribution, with different delay times representing points within that distribution. Each of these delay times will be associated with the same script – one that indicates that the batteries have failed.

Table 3-5. Equivalence of Models Using the “If – Then – Else” Decision Process.

<p><i>Problem description:</i> A model is supposed to execute different instructions depending on the value of a measured temperature T. The model should execute instruction set A if T is less than 50°, instruction set B if T is between 50° and 70°, C if T is between 70° and 90°, D if T is between 90° and 100°, and E if T is greater than 100°.</p>	
<p><i>Explicit</i> instructions – the analyst constructs mutually exclusive logical statement sets to represent the various logical conditions</p>	<p>Equivalent <i>implicit</i> instructions – the model executes instructions only the first set of satisfied logical conditions that it encounters.</p>
<p>If T < 50° Then Do instruction set A Else If T ≥ 50° and T < 70° Do instruction set B Else If T ≥ 70° and T < 90° Do instruction set C Else If T ≥ 90° and T < 100° Do instruction set D Else If T ≥ 100° Do instruction set E</p>	<p>If T < 50° Then Do instruction set A Else If T < 70° Do instruction set B Else If T < 90° Do instruction set C Else If T < 100° Do instruction set D Else Do instruction set E</p>
<p><i>Explanation of the models’ equivalence:</i> The example on the right is equivalent to the one on the left because each of the “≥” operators is <i>implied</i> by the order of the logical cases. Any situation where T < 50°, for example, causes instruction set A to be executed and all further parts of the model to be ignored. Thus, since the logic statement “T < 70°” is ignored when T < 50°, the “T ≥ 50°” part of that logic set in the left hand example is redundant and can be eliminated, as has been done in the right hand example.</p>	

In the jargon of fault tree analysis, the fault conditions described in the previous paragraph both represent “primary” failures (failures that are not caused by outside sources).² A “secondary failure” represents a situation where the component is somehow driven beyond its design margins. An example of a secondary failure is when the system tries to push 200 pounds pressure into a tank that is only designed to withstand 100 pounds. Secondary failures can be handled within the IRM by installing logical conditions that detect violations of the object’s design margins, and executing a script that indicates the component has failed. Similarly, a “command failure” represents a situation where a component has been commanded to operate under conditions where its correct operation will lead to an undesired consequence. An example of a command failure is when the operator of a drawbridge causes the bridge to be raised without first waiting for the cars to get off the bridge. Here the drawbridge functions properly, but at the wrong time. Command faults are naturally handled within the OBEST modeling paradigm because the OBEST model naturally contains each object’s expected behavior. As long as the system model is properly constructed, the object will exhibit that behavior at any time it is stimulated to do so, thus inherently incorporating command faults into the model.

Table 3-6 Improper Model Simplification Using the “If – Then – Else” Decision Process.

<i>Problem description:</i> Same as used in Table 3-5.	
Explicit instructions – the analyst constructs mutually exclusive logical statement sets to represent the various logical conditions.	Nonequivalent instructions – the analyst has wrongly applied simplification to the logic in this model.
<p>If $T < 50^\circ$ Then Do instruction set A Else If $T \geq 70^\circ$ and $T < 90^\circ$ Do instruction set C Else If $T \geq 90^\circ$ and $T < 100^\circ$ Do instruction set D Else If $T \geq 50^\circ$ and $T < 70^\circ$ * Do instruction set B * Else If $T \geq 100^\circ$ Do instruction set E</p>	<p>If $T < 50^\circ$ Then Do instruction set A Else If $T < 90^\circ$ Do instruction set C Else If $T < 100^\circ$ Do instruction set D Else If $T < 70^\circ$ * Do instruction set B * Else Do instruction set E</p>
<p><i>Explanation of the models' nonequivalence:</i> The example on the right is not equivalent to the one on the left because the “*” statements have been moved. The model on the left is still correct because it uses mutually exclusive logical statements. However, the model on the right is incorrect because it now allows temperatures between 50° and 70° to execute instruction set C. Instruction set B is never executed in this incorrect model.</p>	

3.2.3. OBEST Methodology Summary

This section has provided a lot of detail regarding how an OBEST model is *constructed*, but very little information about how an OBEST model actually *works*. Thus, it may still be a little difficult for the reader to see the “big picture” of how the OBEST methodology can be beneficial for infrastructure risk assessment. The next section, which focuses on the OBEST model solution algorithm, should help ease any confusion because the workings of the model are intimately tied to the solution algorithm. However, since this may be many readers’ first exposure to the concepts of object-oriented analysis, we wish to return to the “big picture” and review the basic concepts of the OBEST methodology so that the reader is properly prepared for the next section.

By way of review, the OBEST object model is typical of many common object models in that the information, events, and actions that are related to an object are completely encapsulated within that object and are available to other objects or analysis drivers only through specified

interfaces.^{*} An individual object in an OBEST model begins as a state-based model based on a state transition diagram. This enables the object to respond differently to stimuli when it is in different states. Transitions between states can occur either as a result of conditions external to the object, or because of conditions specified within either the IRM or the DRM. The processes that occur within the object are embodied within the IRMs and DRMs. The IRMs and DRMs vary from state to state within an object. The IRMs and DRMs can embody a wide variety of behaviors, ranging from simple failure models (e.g., an object that simply represents the results of a fault tree) to complex reactive behaviors.[†] The overall object model for an interdependent system is formed by connecting objects that represent individual components according to the influences they exert upon each other in the physical system.

The principal differences between OBEST and the prior Shlaer-Mellor and OPRRA methodologies lies in the treatment of an object's processes or behaviors within a state. OBEST introduces the concept of immediate and delayed response models (IRMs and DRMs), which can be viewed as simple probabilistic transformation functions.[‡] Since these transfer functions are probabilistic, they enable the scenario to branch at appropriate times in the same way as an event tree branches when events occur. The principal difference between the IRM and DRM is that IRM output conditions occur immediately,[§] while DRM output conditions are held in abeyance for an analyst-specified period of time.^{**} In the DRM, the object model is, in essence, saying, "I will remain in my current condition for this length of time, and then my configuration will change to this new set of output conditions." It can also say, "At a specified time, I *might* change to this new set of output conditions, and the probability of that change is this. If that change does not occur, then my output conditions will look like this." The solution algorithm makes use of this information to determine time steps and branching conditions for the overall system scenario simulation.

^{*} In OBEST, the objects communicate with one another by generating events (which can be used to change the state of any object) and by setting attributes (which can be read and reacted to by other objects). An object can only change its own attributes. If an object wants to change another object's attributes, it must request that it do so either by generating an event or by modifying its own attributes in a way that communicates with the other object.

[†] It may even be possible to incorporate objects that exhibit intelligence. Such objects would resemble agents from agent based simulation. However, the study of such objects' feasibility was beyond the scope of this project.

[‡] It is also possible to incorporate "deterministic" components alongside of their probabilistic counterparts because a probabilistic object degenerates to a deterministic object when all of the probabilities therein are either zero or one.

[§] For infrastructure problems, "immediately" is assumed to be on the order of milliseconds to a few minutes.

^{**} The delay may be minutes, hours, or even days. In the current methodology, this delay is specified explicitly by the analyst. In later versions of the modeling methodology, the delay may be calculable by the object model itself.

3.3. Solving an OBEST Model

As we begin the task solving an OBEST model, it helps to begin at a familiar reference point. Begin by thinking of the completed OBEST model as just another tool for simulating the performance and behavior of complex system. A traditional simulation software tool begins with a set of preprogrammed facts it knows about the system and the universe. It augments those facts with user input to set up the specific configuration, situation, and set of conditions to be modeled in this particular analysis. It may consider only the initial conditions of the system, or it may also include definitions for a series of events that are to occur at later times in the scenario. These are often referred to as the initial and boundary conditions of the simulation. A traditional simulation software tool uses the total scenario definition to determine the response of the system to this one unique set of circumstances. If one wishes to explore system behavior over a broad set of conditions, one must define the initial and boundary conditions separately for each scenario to be analyzed, and execute the simulation software many times. By examining system behavior over many discrete sets of conditions, the analyst hopes to come to an adequate understanding of the system's responses to those scenarios that he was not able to simulate.

The parallels between traditional simulation software and an OBEST model are as follows. The OBEST model contains a definition of all of the relevant facts that describe the system and the universe in which it resides. It also contains information needed to set up the specific configuration, situation, and set of conditions to be modeled, and any events that are to be imposed upon the system at later times in the scenario. Thus, the OBEST model contains what would normally be considered the system description, the "laws of nature", and the initial and boundary conditions of the simulation. However, since the OBEST model also contains probabilistic branching information, where a traditional simulation produces one system response to this scenario, the OBEST model will produce descriptions for many probabilistically weighted scenarios. With this background in mind, we now proceed to describe the OBEST model solution methodology.

A high-level view of the OBEST model solution methodology is as follows: the method first applies an analyst-specified set of initial conditions to the system object model, and follows the sequence of events experienced by the system through a series of non-uniform time steps as dictated by the characteristics of the object model. Probabilistic branching of the scenario is possible at each time step, so the method makes use of ideas from the "depth-first" and "best-first" search algorithms⁴⁶ to decide which scenario to analyze first (the other options at each branch point are saved as incomplete paths to be completed later). Since each branch point is probabilistic, the relevant probabilities can be combined to determine the likelihood of the complete scenario. The scenario can be truncated prior to completion either because of a low likelihood of occurrence or because the characteristics of the scenario itself do not satisfy the characteristics of a "useful scenario" as identified by the analyst. When a scenario is completed, the algorithm then retrieves and completes one of the incomplete paths in the same manner. A recursive algorithm is used to exhaustively explore the space of possible scenarios subject to the truncation constraints described above.

The solution process begins with the analyst specifying the initial conditions to the object model. Often, the initial conditions set the system in an equilibrium condition that the analyst then perturbs through the application of initiating events that drive the system out of the initial

equilibrium. At this point, the OBEST analysis supervision algorithm propagates these conditions through the object model by invoking the IRMs for all objects and allowing them to come to a new equilibrium condition. These IRMs can generate events, modify attributes, change states, and otherwise exhibit behaviors that cannot be fully realized within a single application of the IRMs. In addition, feedback effects between objects may require that these models be invoked iteratively under supervision in order to help the system achieve a new equilibrium condition. Thus, the IRMs may need to go through several generations of analysis before this new equilibrium condition is established. Equilibrium is considered to occur when there is no change in the conditions stated by the system model from one generation of IRM execution to the next.

Once equilibrium is achieved, the analysis supervision algorithm queries each object's DRM to determine how long the object can remain in this condition of equilibrium before its internal behavior forces a change of state or attributes (e.g., a generator runs out of fuel or a battery runs dead). The supervision algorithm selects the first such event, steps the analysis forward in time to that event's time of occurrence, and imposes it on the system. At this time, the cycle repeats itself with application of the IRMs over successive generations until another new equilibrium is achieved, then reapplication of the DRMs, and so forth. The process is repeated until pre-defined terminating conditions are satisfied.* This yields a definition for a single possible sequence of events, and the combination of probabilities achieved along the scenario pathway yields the likelihood of that sequence of events.

To this point, the discussion has focused on deterministic models (i.e., IRMs and DRMs that do not make use of probabilistic branching). When only deterministic IRMs and DRMs are triggered, the OBEST methodology behaves exactly like a traditional simulation tool in that it produces only a single system response for each set of input conditions. When probabilistic models are triggered, the solution methodology must eventually follow all of the scenario pathways implied by the probabilistic behavior. If we were to follow the concepts used in Monte Carlo-based discrete event simulation, we would simply draw a random number to select which probabilistic branch was to be taken, and continue the simulation. A better solution, however, comes from the recursive algorithm used to solve large event trees. In this algorithm, the solver maintains a stack of incomplete scenarios that are to be completed at a later time. It also computes the probability of each scenario – complete or incomplete – and stores that with the scenario description. To perform a probabilistic branching operation, the solver selects one branch to carry forward (usually the highest probability branch), and places a complete description of the system (and its history, if required) on the incomplete scenario stack. It then carries the selected scenario on to completion and returns to the stack to resume processing the most recent incomplete scenario. The process continues recursively until the incomplete scenario stack is empty, at which time the solution is complete. By using the most recent incomplete scenario (a “last in – first out” stack), the algorithm minimizes the size required to

* Terminating conditions may be based on elapsed time (“We’re only going to simulate the first 3 days.”), on equilibria (e.g., a situation where there are no other events to simulate), or on logical criteria supplied by the analyst (“We only want to know when this telephone switch dies – everything after that is irrelevant.”).

store the incomplete scenario stack. By selecting the highest probability branch to be solved first, the solution algorithm is similar to the well-known “best-first” variant of the “depth-first” search algorithm from network and optimality analysis.

The OBEST methodology carries with it an inherent assumption that is common among object-oriented modeling techniques: it assumes that all of the objects within the overall system model are operating concurrently. In other words, no object has operational precedence over any other object. If the real-world system involves operational precedence for one object, the analyst must build that behavior into the object model. Given this assumption, it is of particular concern that the solution algorithm must produce results that do not depend on the order in which the individual object models are invoked. While the problem of order-dependence may not be immediately obvious, consider an example in which three objects interact. Object A produces an event that causes object B to change state. In its original state, object B produces an event that affects object C, while in its new state, this event is not produced. If these objects are assessed in the order A-B-C, then object C is unaffected, whereas if the objects are assessed in the order B-A-C, object C is changed. Since we are trying to simulate a situation in which these changes are happening *simultaneously*, this is a very undesirable feature for the model solution algorithm to have.

In order to avoid this order dependence, the OBEST solution methodology enforces a rule that states that the “condition of the universe” (i.e., the conditions of all objects within the model) must be held constant throughout each generation as the object models execute their respective IRMs. To accomplish this, the algorithm proceeds in five steps for each generation.

1. Each object processes the events that were generated during the previous generation and, if necessary, changes its state in response to those events.
2. The system processes any “special” events (e.g., commands to start, stop, or reset an object’s clock, or terminate or truncate the scenario).
3. Each object processes the commands to modify its attribute values that were generated during the previous generation. This sets the conditions for all of the objects that will be used as they evaluate their IRMs.
4. Each object executes its IRM. It determines which set of logical conditions is satisfied by the current system state, and selects the appropriate script to execute. If probabilistic branching occurs, the appropriate incomplete scenarios are placed on the stack for later processing. Any events and “modify attribute value” commands that are generated are placed in lists to be used in the next generation of IRM execution.
5. Once all objects have had the opportunity to execute their IRMs, the system supervisory algorithm examines the lists of events and modify attribute value commands. If the contents of these lists would cause no change to the system’s conditions, then the supervisory algorithm declares that equilibrium has been reached and prepares for DRM execution. Otherwise, the supervisory algorithm prepares for another generation of IRM execution.

This methodology ensures that the problem solution was not influenced by the arbitrary order in which the object models were considered within the solution algorithm. Of course, it is still

possible for objects to respond inappropriately because of changes that occurred in previous generations. That issue must be addressed within the object model itself. The objective of this part of the solution algorithm is to ensure that the analysis results are repeatable even if the supervisory algorithm examines the object models in a different order.

3.3.1. IRM Branching Methodology

The probabilistic branching methodology employed by the OBEST solution method is somewhat more complex than that used in a traditional event tree analysis because of the OBEST assumption that all of the various object models are executing simultaneously. Suppose now that two different object models both reach points within their IRMs where probabilistic branching is required within the same generation. In an event tree analysis, the branching points always occur in a predefined order that is specified by the analyst. This makes implementation of a recursive solution algorithm simple. In the OBEST methodology, one of the key features is the ability for the model to determine the ordering of events “on-the-fly” and even for simultaneous events to occur. This complicates matters for the probabilistic branching methodology.

The simplest branching methodology involves simple combinatorial expansion. Here the algorithm generates every possible combination of branches that occurs within a given generation of the IRM. It then selects one branching combination to continue analyzing, and places remaining combinations on the incomplete scenario stack. Thus, if two objects are each executing binary branches simultaneously, the branching algorithm constructs all four (2×2) possible branching combinations, selects one for further analysis, and places remaining three combinations on the stack. If three objects are each executing three-way branches, though, this combinatorial methodology requires the branching algorithm to construct 27 ($3 \times 3 \times 3$) possible branching combinations, select one for further analysis, and place the remaining 26 combinations on the stack. Obviously, if many objects are branching simultaneously, this combinatorial approach causes the incomplete scenario stack to grow rather large.

While the ultimate result of the branching methodology must be that all possible branches are selected and executed regardless of the order in which objects are considered by the solver, the branching process itself need not be so restrictive. If an algorithm that makes use of object ordering can produce the same results as the combinatorial algorithm – and maintain a much smaller incomplete scenario stack, it would be advantageous to implement this algorithm. Consider a branching methodology that places one item on the incomplete scenario stack for each object that enters a branching situation, rather than each scenario generated within the branching situation. If such an algorithm encountered a situation with three objects each executing three-way branches in the same generation (as described above), this algorithm would place three items on the incomplete scenario stack instead of the 26 required by the combinatorial algorithm. Such an algorithm has been devised, and is implemented in the OBEST solution methodology.

As we begin to describe this new branching methodology, let us consider a simple example where a single object is required to perform five-way branching in a single generation. Assuming that this is the only branching that occurs within this generation, the solution algorithm could choose one of those five branches to be carried forward in the analysis (our method selects the branch with the highest probability, as one would when performing a best-

first search). Let us assume that, for this example, branch one has the highest probability, and branches two through five are to be deferred for later consideration. The solution algorithm could then place the configuration of the overall system object model, as it existed at the time of the probabilistic branching, on the incomplete scenario stack, and simply note that branch one has already been solved. Thus, when this situation is removed from the incomplete scenario stack for further analysis, the solution algorithm notes that this branching operation has not yet been completed. It examines branches two through five, selects the branch with the highest probability (say, branch two), and propagates this branch through the remainder of the analysis while placing the remainder of the branching operation back on the incomplete scenario stack. This time, the item is placed back on the stack after noting that both branches one and two have already been solved. On completion of the scenario for branch two, solution algorithm once again removes this same item from the incomplete scenario stack, notes a branching operation that has not yet been completed and examines branches three through five in the same manner. The last time this item is removed from the incomplete scenario stack, there is only one branch remaining to be taken. This branch is simply propagated through the remainder of the analysis. There is no need to place it back on the incomplete scenario stack since, with the branching process now complete, there are no remaining incomplete scenarios associated with this branching point. In this way, an arbitrarily large branching operation never takes up more than one position on the incomplete scenario stack.

Let us now see how this algorithm might be applied in situations where multiple objects are branching within the same generation. Consider two objects (A and B), each of which is experiencing three-way branching within the same generation. Let us assume that the solution algorithm examines the objects sequentially, object A first. Let us also assume that both objects' branches have been entered in order of decreasing probability, so branch one has the highest probability in both objects. The branching methodology begins by selecting one branch to carry forward from object A. It then places an entry in the incomplete scenario stack describing the object models' current configuration. This stack entry contains a notation that says branch one from object A has already been examined, but branches two and three from object A, as well as all three branches from object B, remain to be examined.

The solution algorithm now begins examining the scenario that includes branch one for object A. The algorithm notes that a second branch point exists within the same generation – that of object B. It selects the most likely branch for object B to be carried forward for further analysis. It then places another entry on the incomplete scenario stack describing the models' current configuration. This second stack entry contains a notation that says all branches from object A have already been examined, and that this scenario makes use of branch one from object A. Furthermore, the notation indicates that branch one from object B has already been examined, but branches two and three from object B remain to be examined. The scenario that incorporates branch one from object A and branch one from object B is then processed to completion.

The solution algorithm now returns to the incomplete scenario stack and processes the remaining to branches from object B (making use of branch one from object A) using the algorithm described previously. The algorithm behaves as though object B were the only object branching during this generation. Thus, three scenarios are generated for branch one of object A. These scenarios represent three possible branches at this branch point from object B.

The algorithm now turns its attention to the second branch from object A. As before, the branching methodology selects one branch to carry forward from object A – in this case, branch two. The algorithm then places an entry on the incomplete scenario stack describing the object models' current configuration, with a notation that says branches one and two from object A have already been examined, but branch three from object A, as well as all three branches from object B, remain to be examined. The solution algorithm now generates a scenario utilizing branch two from object A and branch one from object B while, as before, placing a second entry on the incomplete scenario stack. This entry has a notation that says all branches from object A have already been examined, that this scenario makes use of branch two from object A, and that branch one from object B has already been examined. The algorithm then constructs all three possible scenarios for this branch point for object B while using branch two from object A, as described above. Finally, the same process is repeated to generate all three possible scenarios for object B using branch three from object A. In this way, nine total scenarios are generated for the two three-way branch points using only two entries on the incomplete scenario stack.

The extension of this algorithm to model an arbitrary number of object branch points within the same generation is obvious. It requires only one entry on the incomplete scenario stack for each object that is branching within that generation. This helps minimize the size of the incomplete scenario stack. While it may not be immediately obvious why this is important, the reader should note that each entry on that stack represents a complete copy of the object model for the entire system (as it exists at the branch point in question) plus additional “bookkeeping” information needed for algorithm and scenario management. Thus, an algorithm that minimizes the size of the incomplete scenario stack is important to the computational efficiency of the method. But perhaps the greatest benefit of this branch management scheme comes from the fact that it can be used with only minimal changes to implement DRM branching, whereas the combinatorial branching method could not.

3.3.2. DRM Evaluation and Branching Methodology

Let us now move on to consider DRM branching. DRM branching is more complex than IRM branching because the DRM does not require all aspects of the branching operation to be completed a single time. For example, an object that models the depletion of batteries within a telephone switching facility may use a DRM that incorporates several different delay times to model the distribution of times at which battery depletion may occur. It is also possible that the DRMs in different object models will contain delay times that cause the need for simultaneous branching in the DRMs for two different objects. To make matters worse, it is possible that some branches within a DRM may never actually get executed. Consider the situation where commercial electric power is restored to the telephone switching facility after the first potential battery depletion time but before the last. The DRM branching model must be able to accommodate all of these possibilities.

Let us now examine the behavior of a four branch DRM as the OBEST branching methodology is applied. Let us assume that this DRM has four probabilistic branches, and that the delay times for those for branches are 50, 30, 30, and 60 minutes for branches one through four, respectively. In the DRM branching methodology, the branch's probability does not matter except as a tiebreaker when two branches have the same delay time. In this example, we assume that branch

three has a higher probability than branch two. We also assume, initially, that this object is the only one whose DRM will cause branching between 30 and 60 minutes.

The evaluation of the DRM begins when the OBEST methodology detects that equilibrium has been achieved among the various model objects as they exercise their IRMs. After equilibrium has been achieved, the methodology queries every object in the system object model to determine how long each object can remain in the equilibrium condition that was achieved through the IRMs. These equilibrium durations are embodied in the delay times in the DRM. The object that has the shortest delay time remaining sets the time step for the overall object simulation. The OBEST methodology will step forward in time to the point where that shortest remaining delay time expires, and execute any scripts that are associated with that delay. If, as in this case, the shortest delay is part of a probabilistic branching DRM, the methodology must determine which conditions to propagate through for solution and which to place on the incomplete scenario stack. In this example, this DRM contains two branches with an equal delay of 30 minutes each. The methodology selects the DRM with the shortest delay, and, if more than one branch has the same delay, the one with the highest probability. In this case, we select branch three to be propagated for solution. The algorithm places an entry on the incomplete scenario stack that contains the configuration of the overall system object model as it existed at the beginning of the DRM execution. This stack entry also contains a notation to indicate that branch three has already been solved, thus removing it from consideration when the stack entry is picked up for further processing.

When the scenario associated with branch three is completed, the algorithm examines the entry that was placed on the incomplete scenario stack. It examines the situation and the DRM using exactly the same criteria described above, except that branch three is no longer eligible to be selected. Again, the shortest remaining delay time is 30 minutes, and is associated with branch two. The solution algorithm selects this branch to be propagated through for solution. It also replaces the entry on the incomplete scenario stack with one that contains a notation to indicate that both branches two and three have already been solved. When it determines that branch two is to be propagated for solution, the algorithm takes a time step of 30 minutes, executes the script associated with branch two, and continues processing until the scenario is eventually completed.

The algorithm is now able to re-examine the entry from the incomplete scenario stack. The only remaining branches have delay times of 50 and 60 minutes, representing branches one and four, respectively. Since the shortest remaining delay time is 50 minutes, branch one is selected for solution, while the entry on the incomplete scenario stack is replaced with a notation to indicate that only branch four remains to be solved. To solve branch one, the algorithm takes a time step of 50 minutes, executes the appropriate script, and continues processing until the scenario is completed. Finally the algorithm does a final examination of the entry from the incomplete scenario stack and propagates branch four, with its delay time of 60 minutes, through for final solution. Notice how this branching algorithm is very similar to the IRM branching algorithm, except that time replaces probability as the figure of merit for selecting which branch is to be executed first.

Let us now expand this example to consider how multiple objects can interact through their DRMs. Assume that the object described above represents battery depletion for a telephone switching facility. We will define a second object that represents the recovery of commercial

electric power. Initially, this object has only one entry in its DRM: commercial power is recovered after 40 minutes. Obviously, if commercial power is recovered after 40 minutes, the 50 and 60-minute battery depletion options are no longer relevant. If we assume that the objective of this analysis is to determine whether the telephone switching facility fails because of battery depletion, then each scenario is terminated when either battery depletion or commercial power recovery occurs. Let us examine how the OBEST branching methodology deals with this situation.

The first two scenarios generated by the OBEST solution algorithm for this example problem are identical to those described in previous example: the telephone switching facility experiences battery depletion after 30 minutes, and the scenario is terminated. This leaves us with an entry on the incomplete scenario stack, the details of which are as follows: the entry stores the configuration of the overall system model as it exists before either of this example's two DRMs are executed. It also contains a notation that the two 30-minute branches in the battery depletion model have already been completed, but all other DRM options are available. When the solution algorithm pulls this entry from the incomplete scenario stack, it looks at the entire system model to determine which object has the shortest remaining delay. In this case, the commercial electric power object has the shortest delay (40 minutes). Furthermore, the solution algorithm notes that this is a deterministic DRM because it does not employ probabilistic branching. Because of this, the algorithm does not generate a new entry for the incomplete scenario stack. Rather, it steps forward to the 40-minute time mark and applies the script from the commercial power object, which notes that power has been recovered and terminates the scenario. Note that two of the potential branches representing battery depletion (the 50 and 60-minute branches) were never exercised because they were, in a sense, "overcome by events" and became irrelevant to the overall model. This model behavior realistically represents the behavior of the physical system, whereas a method that required these potential battery depletion branches to be exercised would obtain an unrealistic solution. Furthermore, had we used a combinatorial branching model to represent DRM behavior for battery depletion, it might have been difficult to determine which of these branches should be executed and which should be ignored. Therefore, we see that the OBEST DRM solution algorithm provides an efficient and accurate representation of the race conditions that are so common in infrastructure risk assessment problems.

Consider now a slight variation on the above problem in which there is a reason to continue examining the scenarios even after it is determined whether a power outage actually occurs in the telephone switching facility. The two 30-minute scenarios described above would remain initially unchanged. Then, when commercial power was restored, the telephone switching facility would respond by restoring call processing (after an appropriate delay to represent equipment restoration within the facility). This behavior would have to be included in the object model for the telephone switch. The remaining scenario, where battery depletion never occurred, requires some additional investigation. Recall that, when commercial power restoration occurred, two probabilistic branches in the battery depletion model still remained to be considered. However, the fact that power has now been restored means that these branches are no longer appropriate and should be ignored. This is accomplished by building appropriate logical conditions into the battery depletion DRM. These conditions might state, for example, that if commercial electric power is unavailable, then execute the probabilistic branching described above. Otherwise, do nothing. As soon as electric power is restored, the battery depletion DRM will no longer satisfy the conditions required for this probabilistic branching.

The OBEST solution methodology will “wipe the slate clean” and continue on without further consideration of these two abandoned branches. Since the algorithm keeps track of the total probability associated with each entry on the incomplete scenarios stack, and since that probability has not changed (only the conditions within the model itself have changed), no complicated probabilistic adjustments are required. Such adjustments would have been needed had a combinatorial branching method been used. Rather, the algorithm simply changes the notations associated with the system object model to indicate that the battery depletion model now satisfies a different set of DRM logical conditions.

This brings up an interesting question. What should we do if an object is part way through the delay in its DRM, when the situation changes such that its DRM determines that a different delay would be appropriate? Consider the situation where a camper has gone into the wilderness with a small trailer. This trailer has a bottle that contains a fixed amount of propane fuel. If the weather is warm and he does not have to use the heater in his trailer, the propane will last for 20 days of cooking and miscellaneous uses. However, if the weather is cold and he uses his heater, the propane will last for only eight days. We might represent the camper using his fuel as one object, and the potential weather conditions as a second object in an OBEST model. Assume now that the weather conditions are warm for the first five days, then cold for the next ten days. Since no probabilistic branching is used in either object model, this OBEST model will produce a single scenario that terminates when the camper runs out of fuel. The solution algorithm examines the initial conditions and determines that the propane will last for 20 days. It examines the DRMs for the two objects, and determines that the shortest delay (five days) is associated with the weather model. The solution algorithm takes a step of five days, and executes the “script” that changes the weather conditions to “cold.” We now have a dilemma for the camper using his fuel. We cannot continue using the original 20-day delay time because the logical conditions associated with that delay no longer apply. However, beginning a new eight-day delay (as might be expected based on the newly applicable DRM) is also inappropriate because, after five days of warm weather, the camper has already used 25 percent of his available fuel. The most appropriate solution would be to reduce this new eight-day delay by 25 percent (i.e., to 6 days) to account for the fuel that the camper has already used. In other words, since 25 percent of the initial delay has been “used,” the time available in the new delay should be reduced by that same 25 percent.

The OBEST solution methodology has incorporated this concept of “nonlinear time” in order to allow these types of situations (which are relatively common in infrastructure applications) to be effectively modeled. When the situation within the global object model changes such that the previously applicable set of logical conditions (and its associated delay time) is no longer valid, the algorithm computes the fraction of the previously applicable delay time that has already elapsed, and reduces the newly applicable delay time by that same fraction. The analyst has the option to override this behavior by including the event “reset my clock” in an appropriate IRM script. The “reset my clock” event sets an object’s elapsed delay clock back to zero, or, equivalently, sets the applicable delay time back to its maximum value as dictated by the logical conditions established within the DRM.

Two details remain to be considered with respect to OBEST processing of DRMs. First, what is to be done when several object’s DRM delays expire at the same time? And second, by extension, what happens when the delay times for two probabilistic DRMs expire at the same

time? To answer the first question, we hearken back to the way commands from multiple objects are processed in the IRM algorithm. Recall that, in order to preserve the order independence of IRM model execution, all of the model events, system events, and attribute change commands are placed in “lists” to be executed on completion of the current IRM generation in preparation for the next generation’s processing. A similar technique is used here to accommodate the delays for multiple DRMs expiring at the same time. The algorithm simply executes the script associated with each expiring DRM. In so doing, it places the model events, system events, and attribute change commands in the same lists used by the IRM algorithm. In the same way that each successive object’s IRM commands were added to these lists to provide the impetus for the next generation’s behavior, each successive object’s DRM commands are added to the lists to become “initial conditions” for the next IRM evaluation step.

One might ask what happens if DRMs for two different objects attempt to cause contradictory behaviors to occur. Using the battery depletion example described previously, suppose that commercial electric power had been restored after a delay of 30 minutes instead of 40 minutes as previously assumed. In this case, at the 30 minute mark, one object would be claiming that the telephone switching facility failed due to battery depletion while the other would claim that commercial power had been successfully restored at that same instant. This apparent contradiction would have to be sorted out in the details of the telephone switching facility object model. The analyst can define the appropriate behavior in two different ways. First, the OBEST solution methodology requires the analyst to establish “priorities” within the state transition model. Thus, when an object in one state receives simultaneous events that would cause it to transition to different states, the analyst establishes a priority list to resolve the order in which these events should be considered. If the event and state transition representing power recovery are assigned a higher priority than those representing battery depletion, the simultaneous events will be resolved in favor of power recovery rather than battery depletion.

The second means of resolving simultaneous and contradictory conditions resides in the IRM definition. Recall that the “if – then – else” decision methodology considers the logical cases in IRMs and DRMs in an order specified by the analyst. If an analyst places the logical case to describe power recovery ahead of the logical case to describe switch failure, they are stating that the appropriate behavior during these simultaneous transitions is to continue operation (i.e., power recovery takes precedence over battery depletion). By reversing the order of these logical cases, the analyst holds that the appropriate behavior would be at least a momentary failure in telephone switching operations.

The extension of this DRM evaluation and branching methodology to situations that require simultaneous probabilistic branching among multiple objects is analogous to the branching methodology used when multiple IRM objects branch in the same generation. The algorithm first determines the size of the appropriate time step by obtaining the minimum remaining delay time from each object in the system model. Since two objects require simultaneous probabilistic branching, the algorithm selects the scenario to be propagated forward from the DRM of the first object it encounters. It places an entry on the incomplete scenario stack to indicate that all branches are available except the selected scenario for the first branching object, and that all branches are available for the second branching object. As it propagates the selected scenario forward (still in the DRM), the algorithm comes to the second DRM that requires branching at this time. Again, it selects a scenario to be propagated forward that contains the selected branch

from the first object and, now, the selected branch from the second object. It places a second entry on the incomplete scenario stack to indicate that only the selected branch can be used for the first object, and all but the selected branch are available from the second object. Note that these two entries on the incomplete scenario stack are exactly analogous to the entries described in multiple IRM branching example from the previous section. The remainder of the branching and solution algorithm proceeds according to the pattern established in the multiple IRM branching example, subject to the use of delay as the primary branch selection discriminator as described in the early part of this section. Thus, while the nuances may differ between the IRM and DRM solution algorithms, the basic concepts and applications are the same.

3.3.3. Truncation and Binning

The probabilistic branching rules that are characteristic of the OBEST modeling methodology can cause the solution algorithm to produce a very large number of probabilistic scenarios for each set of initial conditions analyzed. The OBEST methodology provides two methods to help the analyst cope with this explosion: scenario truncation and binning. Truncations allows the analyst to instruct the solution algorithm to stop evaluating a scenario before it reaches completion and throw its results away if it is “unimportant” to the desired overall results. Binning allows the analyst to combine the scenarios that are of interest into groups of scenarios, or “bins,” where all scenarios in the bin are “similar” to each other based on binning conditions supplied by the analyst. The binning process reduces what may be hundreds of thousands or even millions of possible scenarios generated by the OBEST solution algorithm into a few dozens or hundreds of bins that can be reasonably examined by a human analyst.

The truncation algorithm used in the OBEST solution algorithm allows the user to truncate a scenario based on several different criteria:

1. The algorithm can be instructed to throw away scenarios whose probability falls below some user-specified cutoff value,
2. The analyst can include explicit instructions in IRM or DRM scripts to indicate that any scenario executing that script is, by definition, not of interest the analyst and, thus, should be thrown away, and
3. The analyst could include explicit instructions in the IRM or DRM scripts to indicate that any scenario executing that script is, by definition, of special interest to the analyst and, thus, should be kept. Any scenario that has not been so terminated when an analyst specified “stop time” is reached is assumed not to be of interest and is thrown away (this truncation condition is not yet implemented in the OBEST demonstration software).

While other truncation and termination conditions could be imagined, these form an extremely powerful facility to enable the analyst to filter out inappropriate results. This is especially true when one combines the “truncate” instruction with the power inherent in the IRM and DRM modeling methodology.

The binning process provides a mechanism for the analyst to use in aggregating similar scenarios into larger groups to ease the manual analysis task that follows solution of the OBEST model.

The method to be implemented in the OBEST methodology uses a method similar to that employed in Sandia's existing SETAC event tree analysis code system. Briefly stated, the SETAC binning methodology asks the analyst to define a series of topics or "dimensions" that are to be used in the scenario aggregation process. For each dimension, the analyst defines a set of legal categories or "values"* that are to be used to characterize the scenario. The binning process then selects exactly one value for *each* dimension in order to summarize the characteristics of that scenario. This unique combination of values is called a bin. The solution methodology combines all scenarios that have the exact same combination of values for *all* dimensions into a single aggregate bin.

The concept of binning can be clarified by the use of a non-OBEST example. Consider the task of categorizing a series of objects. One might start by defining three topics (dimensions) that can be used in this categorization: size, shape, and color. The legal categories (values) for size might be small, medium, large, and huge. The values for shape may be pyramid, rod, sphere, or box, while the values for color may be red, violet, blue, green, yellow, orange, white, black, and gray. Recall that a bin represents one unique combination of values – one value for each dimension. Thus, one bin might describe small red boxes, while another one would describe huge green pyramids, or large orange rods, or medium gray spheres. The binning method keeps separate those bins that may share values from one or two dimensions but are not the same for *all* dimensions. Thus, it tracks large orange rods separately from large orange boxes or huge orange rods or large blue rods. If one wishes to obtain the number of rods, the number of violet objects, or the number of objects that are not small, one must aggregate the bins over the appropriate dimensions to produce these summaries.

While this example uses only three dimensions, one can imagine a binning scheme that uses many different dimensions to describe and summarize the paths generated by solving an OBEST model. SETAC analyses have routinely used binning definitions that contain 10 or more dimensions. We anticipate that similar binning models will be used to summarize OBEST analyses.

The SETAC binning methodology is designed around an event tree analysis paradigm. Thus, its processing characteristics are tailored to look back only at the events that have either occurred or failed to occur in a given scenario. Event tree analysis does not explicitly consider time (except to the extent that it is incorporated in the discrete events that define the event tree model). Thus, the SETAC binning methodology does not allow the analyst to summarize over dimensions such as, "Within which time regime did this event occur?" or "In which order did these three events occur?" These concepts are incorporated in the OBEST binning methodology. OBEST also allows the analyst to specify combinations of attribute values to be used as bin or value definition criteria using an "if – then – else" decision methodology that closely mimics that used in IRM and DRM definition. Finally, since the IRM and DRM definitions do not allow the analyst to

* Note that in the SETAC software, the entities known as values here are called attributes. The terminology was changed for the OBEST methodology in order to avoid confusion between this concept and the attributes described in the OBEST object methodology.

consider an object's state, and since an analyst may be very interested to know whether an object ever entered a particular state over the course of the scenario, this concept is also included in the OBEST binning methodology.

3.4. Compatibility With Parallel Processing

The OBEST solution methodology is, at its heart, fundamentally a recursion method. Through recursion, the method engages in the systematic exploration of all possible branches within a tree-based logical structure. Such problems and solution methods can easily be adapted to make efficient use of parallel processing technology. Note that each entry on the incomplete scenario stack represents a self-contained independent starting point for a new portion of the scenario tree. Thus, a very simple method to parallelize the solution algorithm is to have one processor manage the incomplete scenario stack and pass out stack entries to separate processors for evaluation. As each processor completes a path, it can perform the binning assessment internally and pass back to this supervisory processor only the characteristics and probability for the bin satisfied by the scenario. While this may not make optimum use of the processors on a massively parallel machine, it would dramatically increase the solution speed for an OBEST model.

Another aspect of modern PRA that is easily parallelized is, in many cases, the uncertainty analysis. The use of Monte Carlo and Latin Hypercube sampling (LHS) methods²⁸ to perform such analyses is common. Here, the analyst samples various PRA model parameters prior to the model being executed. The individual samples (observations) are used to create a number of independent problem definitions that must be solved such that one complete set of results is generated for each observation. It is extremely simple to parallelize a process such as this by placing one independent problem definition and solution algorithm (even a non-parallel solution algorithm) on each processor, and collecting the results for output once all of the analyses have been completed. Given that the OBEST solution algorithm can be parallelized itself in addition to parallelization of the uncertainty analysis methodology, the potential is very good for seeing dramatic increases in OBEST solution speed under parallel processing. It may even be possible to see speed improvements that are nearly linear in the number of processors for relatively large OBEST problems and their associated uncertainty analyses.

3.5. Comparisons With Other Methodologies

The various methodological descriptions in this paper should give the reader some insights into the ways that the OBEST methodology both compares to and complements other infrastructure analysis methodologies described in Chapter 1. OBEST inherently characterizes a very large scenario space in a probabilistic manner. This differs from the Aspen model, as well as from traditional network analysis models, which provide a detailed examination or simulation of a single scenario for each model execution. A probabilistic wrapper would have to be placed around these methods in order to achieve the effects embodied in OBEST. In addition, the Aspen model requires vast computational resources to obtain even a single simulation result, although the economic consequences that it simulates are well beyond the scope of the OBEST models that are now being contemplated.

The OBEST methodology also differs from the discrete event simulation modeling methodology that was characterized for the LASEP-T and SimMod-Pro models. While both methods perform a probabilistic characterization of a large event scenario space, LASEP-T and SimMod-Pro are fundamentally a Monte Carlo methodology. One of discrete event simulation's fundamental characteristics is that all event scenarios that it discovers are, by definition, equally likely. This means that one must simulate enormous numbers of scenarios in order to facilitate the discovery of rare but high-consequence scenarios. This can involve great computational expense. The OBEST methodology, by comparison, computes an estimate of the likelihood of each scenario as it is discovered. Thus, the OBEST methodology will facilitate greater computational efficiency for those problems to which it is suited. Note that some problems may still require the complexity of a full Monte Carlo discrete event simulation model. The problems for which LASEP-T was designed may be a good case in point. After all, the scenarios involved in launch-phase accident assessment may not fit the paradigm of immediate response/delayed response modeling that is inherent in the OBEST methodology.

The OBEST methodology can be viewed as a direct extension to traditional PRA models such as event trees, hybrid PRA-simulation models such as DYLAM and DETAM, as well as to traditional object-oriented analysis models (specifically, the Shlaer-Mellor and OPRRA methods described in Section 3.1). The OBEST methodology provides a mechanism for analysis of event scenarios for systems with inherent feedback loops (an area where event tree analysis, influence diagrams, and many other traditional PRA methods fail). It does so without the time step, simultaneity, time scale, and branching limitations inherent to DYLAM and DETAM. It also provides a mechanism for the use of object-oriented models in the area of time-dependent surety analysis (an area that was not fully explored in the OPRRA methodology). The OBEST methodology will not be the most appropriate paradigm for all such analyses, however. In some cases, the time required to develop the object models may be more than would be required to perform a traditional event tree analysis. In addition, the paradigm of immediate response/delayed response modeling may not be appropriate for certain object-oriented surety analysis problems. Thus, the OBEST methodology can best be viewed as a member of a risk or surety analyst's toolbox — being applied only to the analysis of appropriate problems.

The comparison between OBEST and the enumerative risk assessment models is not as straightforward. Methods such as FMEA and HAZOP are applied to systems as a “first stage” of risk assessment. They examine only one or two steps in a hazard scenario but do not simulate longer series of events to their ultimate completion. This represents a significant difference from the OBEST methodology because OBEST is designed to simulate the system's behavior through an entire series of events until consequences are determined. An OBEST model could be constructed that would systematically apply each failure mode to a system model and determine which failure modes lead to consequences in a manner similar to FMEA or HAZOP, but it is unclear whether there is any real advantage to this approach because of the simplicity of these more common methods.

When the OBEST methodology is compared to a Markov model, one comes to the same conclusions stated in Section 2.6 (where Markov models were compared to other probabilistic risk assessment methods). A Markov model can identify the probability that a system will exist in a particular state as a function of time (OBEST cannot). However, a Markov model cannot easily produce the path or scenario information that is so useful to the risk analyst. In addition,

while Markov modeling often suffers from a state explosion when large systems are analyzed, OBEST does not. OBEST *may* suffer from a path or scenario explosion, but whereas the OBEST solution methodology can cope with such a path explosion, the *analyst* must cope with the state explosion because the explosion is in the model rather than the solution.

Comparing OBEST with traditional network analysis methods is a little like comparing apples and oranges. OBEST models discrete series of events, while a network analysis method is designed to assess the flows, capacities, and/or stability of a network under a pre-defined set of conditions. The network analysis tool is inherently aware of the conditions that exist throughout the network, while an object model attempts to encapsulate the behavior of each individual component into a separate package. The two methodologies complement one another in this way. An ideal solution might be to couple a network analysis tool to an OBEST analysis. In so doing, one would use the OBEST model to model the series of events that are impressed on the network, and use a traditional network analysis tool to model the characteristics of the network as the OBEST model determines that network boundary conditions change. If the network analysis tool determined that the network could not function under the conditions imposed by the OBEST model, the network analysis tool could, say, identify overloaded and/or failed components for the OBEST model, which would then propagate their effects through the remainder of the model. In this way, each modeling methodology performs the portions of the analysis for which it is most capable.*

The OBEST methodology also complements the existing suite of Sandia infrastructure and analysis tools: Aspen[†] and the Indications and Warnings System. While Aspen provides a detailed look at the economic response of a region to postulated infrastructure disruptions, it does not provide a workable engine for the simulation of large numbers of such scenarios, and is incapable of a probabilistic exploration of a large scenario space without an external driver. Even then, Aspen's computational requirements would be enormous. At the opposite end of the spectrum, the Indications and Warnings System, being a quasi-quantitative assessment tool, is capable of providing insight into large numbers of scenarios (provided a suitable probabilistic driver is employed). However, the Indications and Warnings System is not designed to provide a simulation-like timeline of the potential scenarios that might occur. The OBEST methodology fits squarely between these two existing methods. Its simulation capabilities are not as detailed as those of Aspen, but are more rigorous and extensive than those of the Indications and Warnings System. OBEST is computationally more intensive than the Indications and Warnings System, so it may not be as useful for "real-time" decision-making. However, OBEST will run much more quickly than Aspen, making it suitable for exploration of large scenario spaces and

* Similar statements could also be made about DYLAM and DETAM.

[†] While this paragraph singles out Aspen, its conclusions are applicable to all traditional simulation methods that might be used for infrastructure analysis to the extent that their computational demands do not allow them to be exercised to evaluate large numbers of scenarios. Those that do not have these stringent computational requirements are similar to the discrete event simulation models described previously in that they must be driven by a Monte Carlo "front end" to probabilistically explore a scenario space if they are to be fairly compared to OBEST.

for risk studies, where a broad perspective of the entire set of possible scenarios is important. The fact that OBEST explores the scenario space probabilistically makes it especially well suited to risk studies.

The potential synergies between Aspen and the OBEST methodology should not be understated. One needs look no further than the area of nuclear reactor risk analysis to understand that a variety of models operating at different levels of detail provide complementary information, all of which is important to the risk analyst.^{47, 48, 49, 50, 51, 52, 53, 54} The detailed models provide insight and guidance for the development of simplified, risk-appropriate models. The risk analyst's exercising of the simplified models over a very broad scenario space raises questions that can often only be credibly resolved using more detailed models. In this way, the detailed models influence the formation of the simplified models, while the simplified models help direct the scenarios to be explored using the limited number of available evaluations of the detailed models.

3.6. *Potential OBEST Applications*

The OBEST methodology was originally developed to enable risk assessment models to be constructed for interdependent infrastructure systems. One might imagine an infrastructure problem that involves electric power, telecommunications, water and emergency services. If electric power fails, there will likely be a significant delay before telecommunications, water, and/or emergency services are effected because each has some backup capacity. However, if a major power failure lasts for an extended period of time, the backup batteries at the telecommunications center and emergency services providers may become exhausted and the reserve water supply may run out. The lack of water and/or telecommunications could, in turn, make it more difficult for the electric power provider to restore service by impairing the coordination of restoration work. Thus, the delay in power restoration is exacerbated by failures in telecommunications and/or water. It should be obvious by now that this situation can be modeled very well using the OBEST methodology. The chief hindrance to the extensive modeling of infrastructure problems using OBEST occurs when the risk model must look for cascading failures in the infrastructure networks. Section 3.7 describes a potential extension to the OBEST methodology that helps alleviate this problem.

While the OBEST methodology was originally developed for the assessment of infrastructure interdependencies, its potential areas of application are much broader. As the methodology has been discussed with other analysts, applications have been considered in areas including aviation safety, physical security, nuclear reactor safety, information surety, and weapon safety. OBEST may, in fact, be as generally applicable as the event tree analysis methodology that is currently popular among risk analysts. Chapter 5 provides a number of OBEST example problems – some of which are derived from these domains. However, the remaining paragraphs in this section provide some ideas of basic problems that may be amenable to OBEST solution.

Probably the greatest use of very large event trees in PRA occurs in the field of nuclear reactor safety. Most modern reactor PRA studies model the phenomenology associated with an accident's progression from core damage through possible radiological release using an event tree that contains between 50 and 150 multi-branch events resulting in literally millions of possible event sequences (paths). These event tree models are notoriously difficult to construct,

examine, assure, and explain. Furthermore, since it is the *analyst* that must distill the system behavior to obtain the event tree model, a change in plant design or an improved understanding of some aspect of accident progression methodology must be manually incorporated in and propagated through the model. This is an error-prone and labor-intensive process. The OBEST methodology could replace event trees in this application since OBEST is in many ways similar to the event tree methodology that is commonly in use. However, OBEST could improve the situation because OBEST modularizes the model into objects, so whereas the analyst must manually propagate changes through the entire event tree model, the changes to an OBEST model would be very localized and, hence, easier to develop and assure. In fact, this compartmentalization should even make the original model easier to develop. Thus, OBEST could provide significant enhancements to current nuclear safety PRA studies.

The area of physical security provides opportunities for OBEST in two major areas. First, some physical security studies have made extensive use of large event trees. For the reasons described above, OBEST could have an immediate impact on these studies. Second, an important part of some physical security studies is “force-on-force” experiments. In some cases, these involve actual people acting out security exercises, while in others, human players engage in similar exercises in a computerized environment. It is likely that the concepts of immediate and delayed response models from OBEST could provide a mechanism to automate some of these simulated security exercises and provide a probabilistic basis to some classes of force-on-force experiments. Generating OBEST models to represent the training and doctrine of both teams (as well as the facility being defended) would not be an easy task, but providing a probabilistic basis and a more thorough and systematic investigation of potential scenarios may be worth the effort. Information surety may be able to benefit from OBEST for similar reasons.

Another area that has made extensive use of large event tree models is the assurance of nuclear weapon safety in transportation accident scenarios. Here, the event tree model is used to define the characteristics of the transportation (including the impact severity, fire characteristics, etc.). The OBEST methodology may provide an opportunity to generate models that are easier to assure, modify, and explain to reviewers than are the current large event tree models.

Aviation safety is an area that has already seen active research related to the OBEST methodology. Early models have included the reformulation of a large event tree aircraft icing analysis, a simple model to assess the closeness of approach between two planes during a runway incursion scenario, and a model to determine the likelihood of a taxiing aircraft to end up on an active runway without permission. The latter model considers command errors from the traffic controller, misunderstandings and memory lapses on the part of the flight crew, responses to visual cues, environmental factors, and the ability of the traffic controller to recognize and command a stop to a dangerous situation. In this model, DRM delays are used to represent the time it takes an aircraft to traverse a particular portion of airport taxiway, as well as the time it takes for a controller to identify and respond to a developing situation where an aircraft is not where the controller believes it should be. Research in this area will likely continue and provide valuable aviation safety insights.

A final (and seemingly disconnected) area of applicability of the OBEST methodology is in the area of electric circuit analysis. Specifically, analysts look at situations where a fire erupts in the area of electric cables. In the presence of fires, these cables can exhibit various types of open

and short circuits. The behavior of the system is dependent not only on *which* open and short circuits occur, but also on the *order* in which those circuit faults occur. OBEST can inject those faults into a circuit model probabilistically and compare the response of the actual circuit with that of an “ideal” circuit (one that behaves as would be expected in the absence of the fire conditions). This comparison can then be categorized to identify the potential for these faults to cause the circuit to behave in abnormal or dangerous ways.

The OBEST methodology also shows promise for compatibility with several optimization and uncertainty analysis techniques. Sandia’s current uncertainty analysis techniques for large event trees should prove immediately transferable to the OBEST process. As for optimization, Sandia is already making use of optimization techniques using system cut set expressions to measure reliability as one portion of a utility metric. In a sense, the system cut set expression represents the detailed investigation of a large potential scenario space, and its quantification summarizes that scenario space for the utility metric. In the same way, the scenario space explored by the OBEST methodology can be summarized in many different ways to yield important information for inclusion in an optimality utility metric. While the re-quantification of an OBEST model is not nearly so simple as the re-quantification of a cut set expression, the basic applicability does exist, and it may ultimately be possible to provide shortcut requantification methods for an OBEST model once an initial solution is found. This may provide greater applicability of the OBEST method to optimality analyses should re-resolution of the model at each stage of the optimality calculation prove infeasible.

3.7. OBEST Limitations and Possible Extensions

In its original form, the OBEST methodology contains a number of inherent limitations that, if removed, would make the method more widely applicable. Chief among these limitations is that the only mechanisms available to determine proper branching are time, probability, and logical statements based on object attributes. The ability to operate on different types of quantities and perform “side calculations,” as is currently done in Sandia’s SETAC⁴⁵ event tree analysis software, would make OBEST dramatically more powerful and flexible. For those who are unfamiliar with SETAC, the side calculations that it facilitates are based on real variables.* The model can set the value of a variable based on the a result of any event branching operation in much the same way that attribute values are set in an OBEST IRM or DRM script. These variables can then be subjected to add, subtract and multiply operations with other variables. They can also be used to influence event tree branching decisions when they are compared with threshold values and probability distributions. The utility of this type of variable is illustrated as follows: the current methodology requires the analyst to represent quantities like the remaining capacity of a battery string or fuel tank through the surrogate variable of time. The capacities are converted into depletion times based upon the draw-down rate imposed by the system being modeled. Thus, if one changes the draw-down rate for that quantity, the OBEST model gets messy. If one could represent the capacity, the draw-down rate, and the landmark value at which

* Real, or continuous variables, as opposed to discrete variables such as integers or Boolean values.

the next action occurs as continuous variables, then the OBEST model could directly compute the delay time until the landmark value is reached and implement the DRM conditions as described previously. The advantage comes when the quantity is partially depleted and the flow rate changes. A variable-based computation method can simply update the remaining capacity and compute a *new* delay time based on the new depletion rate. One could also change the landmark values if that were necessary. These factors could immediately be used to compute an accurate representation of the time-dependent behavior of the model, whereas obtaining accurate behavior once these quantities have been converted into the surrogate time variable is difficult. The ability to use continuous variables into OBEST models would enable a wide variety of systems to be modeled that can otherwise only be crudely approximated. The OBEST method should incorporate variable capabilities that are at least as powerful as those in SETAC and, if feasible, dramatically expanded.* While it would be desirable to restrict the performance of variables such that OBEST is still able to compute the *endpoint* of a DRM time step from the *beginning* of that DRM evaluation, one could envision a capability where even simple differential equations could be incorporated into the OBEST response models through the use of variables.

A second fundamental limitation of the OBEST methodology is that it is inherently based on discrete mathematics. Object behaviors are discretized into states that call discrete IRM and DRM models. Attributes that describe the characteristics of an object can only be represented as discrete values. This requires the analyst to manually choose the landmark values to form the breakpoints within the discretization of what is truly a continuous variable. With the exception of the variables described above, it is difficult to maintain the tree-like behavior of the OBEST methodology while removing this limitation. Other modeling methods such as DYLAN and DETAM³³ may be more suited to problems that cannot be effectively modeled within the constraints imposed by variables once they are implemented in OBEST.

Another limitation of the OBEST methodology is that, as currently implemented, it can be difficult to represent the global behavior of a network using the limited facilities and compartmentalization[†] required by OBEST. There are two possible methods by which an OBEST model can incorporate network effects such as transients within electric power grids or telecommunications networks. The first method involves incorporating into each network element's OBEST object the intelligence to enable it to negotiate with its peers, follow network transients and reach network equilibrium. This method requires extremely complicated objects,

* These types of variables could also be used in IRM branching, where the analyst might indicate that if a particular variable were greater than a certain threshold, then the object would provide an immediate response (such as a tank rupturing or the issuance of a system shutdown command).

† Compartmentalization requires that an object's behavior be completely described within the object itself. Where the behaviors of two or more objects are highly interdependent, as in the case of the components within an electric power network, this strongly interdependent behavior can be very difficult to model using a compartmentalized model. These behaviors are often dependent on small changes in parameter or attribute values, and the discretization of attribute values required by OBEST compounds the difficulty in representing these strongly interdependent systems.

and it will be difficult to ensure that their collective behavior appropriately matches observed real-world network transient and equilibrium conditions. The second method involves creating an extremely simple OBEST object for each network element, and using it as a mechanism to transfer information about the global system state into a traditional network analysis model. Here the OBEST network element model acts principally as an intermediary between the more global OBEST model and the traditional network model of the particular infrastructure to which it belongs. The network model itself follows the transient, imposes consistency, and develops network equilibrium conditions. It then passes this information back to the OBEST objects for the elements of its network, which in turn reset their internal states and attributes to match those found by the network model. Other objects within the OBEST model can then use this information to assess the interactions between this infrastructure and others that are part of the overall system model. This methodology enables the OBEST method and traditional network analysis tools to work in tandem — each performing the portions of the analysis to which they are most suited. A similar method could be used to link an OBEST model to other types of simulation models were that to be found beneficial. Such a linkage could become even more powerful and flexible if it were implemented through the use of continuous variables (as described previously) to enable exact values to be transmitted to the simulation model rather than just nominal values associated with discrete attributes.

Yet another limitation of the OBEST method over traditional ETA methods involves the relationship between fault trees and the event progression model. Most major ETA codes provide a mechanism by which the analyst can specify that a branch probability within the model is really the result of a system failure that has been modeled elsewhere using a fault tree. Since there may be many such branches within the model, and since these fault trees may be interdependent, the event tree software must maintain the relationships between these fault trees and provide for their solution such that the commonalties among fault tree events are properly accounted for in the branch probabilities used in the OBEST event sequence computation.^{2, 47}

Finally, the OBEST methodology can be viewed as *being* a limitation on the previously developed OPRRA methodology. OPRRA was developed to encompass as many traditional PRA methods as possible into a single object-based method. The OBEST methodology significantly limits the action language in OPRRA by moving totally away from Shlaer and Mellor's data flow diagram concept and instituting the concept of immediate and delayed response models. While this limitation excludes traditional deductive risk assessment models from the OBEST paradigm, it is essential to the realization of the extensions OBEST presents to the inductive analysis methods such as ETA. Since the IRM/DRM concept is fundamental to the gains claimed by the OBEST methodology, it is doubtful that extensions to overcome these limitations will be developed any time soon.

3.8. Summary

This chapter has sought to provide the reader with all of the fundamental underpinnings required to construct and solve OBEST models. It has described the object-oriented methods (Shlaer and Mellor, OPRRA) that form the heritage from which OBEST was drawn, as well as those aspects that make OBEST unique from these prior methods. OBEST shares with these methods the concepts of a state space-based approach, where each state represents a different body of system behavior, and where transitions between those states occur as a result of events that can be

generated either internal to or external to the object. However, while the prior methods embodied their specific behavioral models in terms of action data flow diagrams, OBEST embodies its behavioral models in immediate and delayed response models (IRMs and DRMs). This enables OBEST to specifically model systems that are characterized by long periods of quasi-equilibrium, peppered with short periods of potentially intense activity and behavioral changes. This modeling method was motivated by the study of infrastructure systems, but has proven much more generally applicable as described in Section 3.6.

This chapter has also described the algorithm that has initially been proposed and implemented in software for solving these OBEST models. The similarities and differences between IRM and DRM branching were described, as was the treatment of the system and object clocks under different scenarios. The applicability of parallel processing to this solution method was also discussed. Finally, the characteristics of both the modeling method and the solution algorithm were compared with the list of desirable characteristics from Section 2.11. Limitations of the methodology were noted, and potential extensions were proposed to deal with some of those limitations.

The remainder of this report is dedicated to the implementation and applications of the OBEST method. A brief overview of the developmental software that implements OBEST is presented in Chapter 4, while a number of OBEST example problems are detailed in Chapter 5. Together, these sections should provide the reader with additional perspective through the use of concrete examples that illustrate the method, its uses, and its limitations.

This Page Intentionally Blank

4. OBEST Software Implementation

4.1. Software Overview

The models created using the OBEST modeling methodology increase in complexity rapidly as objects, states, and attributes are included. Thus, it is impractical to solve anything but the simplest models by hand. Rather, the methodology demands the use of specialized software to track the varied and potentially competing behaviors and time delays associated with a realistic OBEST model. For this reason, this project also created a software package to solve those models.

The OBEST model solution software (hereafter referred to as the OBEST software, or simply “the software”) was developed over a period of more than one year. During that time, the OBEST methodology itself was evolving through several significant changes as sample problems were worked by hand to gain insights into the nuances and potential problems posed by the conventions and rules proposed for this methodology. For these reasons and many others, this software should be viewed as demonstration software rather than production-grade software. We hope that later projects that make use of the OBEST methodology will be able to fund upgrades to the computational kernel (as discussed in Chapter 3) and completion of those aspects of the user interface that had to be neglected in order to ensure the proper functioning of the computational kernel.

The OBEST software was written using Microsoft Visual Basic for use on an IBM-compatible personal computer running a 32-bit Microsoft Windows operating system (95, 98, Me, NT 4, or 2000). Installation requirements are modest by current computing standards – less than 10 MB of free hard disk space are required. The software generally runs well on computers having at least 64 MB of RAM and a Pentium-class processor. The software has no special graphics, sound, or other hardware requirements. The software stores its model input and results as a database file using the Microsoft Access format. Thus, in the unlikely event that the model itself should become corrupted by the OBEST software, the adventurous user can edit the model file directly using Microsoft Access.

The OBEST software will be described in four main phases. First we describe those aspects of the software that are used for describing the basic characteristics of the object model: naming objects, states, state transitions, attributes, and discrete attribute values. After that, we describe those parts of the software through which the user will describe and enter the IRMs and DRMs, with their associated logical cases and scripts. Third, we describe the parts of the software that are used to specify the logical conditions that will be used for binning the results of the OBEST analysis. Finally, we describe the parts of the software that enable the user to set a model’s initial conditions, execute its solution, and view the results.

4.2. Entering a Basic Object Model

When a user first starts the OBEST software, the screen shown in Figure 4-1 is displayed. This screen enables the user to:

- add and delete objects from the OBEST model (using the “Add New Object” and “Delete Object” buttons),
- add details to the object models (using the “Object Definition” button),
- specify the model’s initial conditions (using the “Initial Conditions” button),
- develop the logical conditions by which the OBEST model results will be binned (using the “Binning Conditions” button), and
- invoke the computational kernel to begin solution of the OBEST model itself (using the “Run Scenario” button).

The two other buttons on this screen (“Object Relationships” and “Termination Conditions”) are currently nonfunctional. In this section, we will focus on the functionality provided by the Add New Object, Delete Object, and Object Definition buttons (parts of the functionality provided by the Object Definition button will also be described in the next section).

Figure 4-1. OBEST Software Main Screen.

When the analyst begins using the OBEST software, the OBEST model initially contains no objects. The list box in the Object Models frame in Figure 4-1 is empty. The analyst begins populating the OBEST model by entering a name and description for each object to be used in

the OBEST model. This is done using the Add New Object button. The object description that is requested for each new object can be thought of as a note or memo field to remind the analyst of the intended characteristics of this object. New objects can be entered into the software all at once (and populated simultaneously), or the analyst may wish to create a single object, and specify all of its model characteristics, before entering information for a subsequent object. In practice, we have found it beneficial to enter the names, states, state transitions, attribute names, and attribute values all at once for as many objects as possible very early in the model development process. This is because the IRMs and DRMs for any object may well depend upon the particular attribute values associated with some other object. In order to ensure consistency within the final OBEST model, the software will not allow the analyst to use object or attribute names that have not already been defined within the model. Thus, it is helpful to define these high-level characteristics of each object before proceeding on with IRM and DRM input for any of the objects.

Another function that is performed directly using the screen shown in Figure 4-1 is that of deleting an object from the OBEST model. To delete an object, the analyst selects the name of the object to be deleted from the list of model objects found in the Object Models frame, and then presses the Delete Object button. This removes the object and all of its states, state transitions, attribute names, attribute values, IRMs and DRMs, and scripts from the OBEST model. The only things that may remain after an object is deleted are those elements of the object model that are not associated with one particular object: events names, and logical case definitions (these will be discussed in a later section).

The image shows a software dialog box titled "Object Definition". It is divided into three main sections. The left section, labeled "Objects", contains a list box with three items: "Aircraft", "Controller", and "Environment", with "Aircraft" selected. The middle section, labeled "Object Description", contains several input fields: a text box with "Incurring Aircraft", a dropdown menu labeled "Initial State" with "Waiting" selected, a checked checkbox labeled "Clock On", and a text box labeled "Clock" with the value "0". The right section contains five buttons stacked vertically: "State Definition", "State Transition Table", "Attribute Definition", "Immediate Response Model", and "Delayed Response Model". At the bottom of the dialog are "OK" and "Cancel" buttons.

Figure 4-2. Object Definition Screen.

Once the analyst has entered names and descriptions for the objects in their model, the next step is to define the states, state transitions, attribute names and attribute values for each object. To begin this process, the analyst presses the Object Definition button. This action causes the software to bring up the Object Definition screen, as shown in Figure 4-2. This screen allows the analyst to directly specify the default initial conditions for each object, and also serves as a navigation center to enable the analyst to examine and define the various aspects of the object model. In this section, we will focus on the functions provided by the State Definition, State Transition Table, and Attribute Definition buttons. The functions provided by the Immediate Response Model and Delayed Response Model buttons will be described in the next section.

Figure 4-3. Entering Object States.

To define the states associated with one or more objects, the analyst presses the State Definition button. This causes the software to bring up the State Definition dialog box shown in Figure 4-3. This dialog enables the analyst to specify the name and description for each state associated with an object. As before, the description field provides the analyst with an opportunity to making note or memo reminding them of the specific behaviors that this state is intended to represent. The list box of the top of this dialog contains a list of all objects that currently exist in this OBEST model. When the analyst selects one of these objects, software responds by populating the text box of the bottom of this dialog with all of the states that have previously been defined

for this object. For a new object, this list will be empty. The analyst can add a new state for this object by entering a name and description in the text boxes provided, and pressing the Add State button. This action enters the new state into the model and adds it to the list of states at the bottom of the form. The analyst can repeat this process for this same object to add multiple states, or add states to a different object by selecting that object's name from the list of the top of the form. When all of the desired states have been added, the analyst presses the OK button to return to the Object Definition screen.

When the analyst selects an object from the Object Name list at the top of the dialog, the software responds by listing all known states for that object at the bottom of the form. If the analyst selects one of those states, its name and description are placed in the text boxes on the form. The analyst can review this information by scrolling through the text boxes, and make modifications as necessary. If the analyst wishes to delete this state from the object, this is done using the Delete State button.

When the analyst returns to the Object Definitions (Figure 4-2) screen after defining the states for each object, they can now enter the basic initial condition information for each object. This consists of specifying the initial state and the initial clock conditions for each object. To specify this information, the analyst first selects the name of the object from the list in the upper left corner of the Object Definitions screen. The software responds by displaying the state description and any previously entered initial condition information for this object. To specify the initial state for this object, the analyst will simply select one of the previously defined states from the Initial State drop-down menu. This process is repeated for each object in the OBEST model. While this menu also includes a Cancel button, that button does not yet function properly. Thus, any changes to the model initial conditions are permanently entered into the OBEST model database. Mistakes are corrected by re-entering the data in the original manner.

The other initial conditions that are set from the Object Definitions screen have to do with the initial conditions for the object's clock. In general, the initial value for each object's clock will be zero, although there may be special situations where an analyst wishes to set a clock's initial value to some nonzero value. To do so, the analyst enters the appropriate value in the Clock text box. In addition to setting an initial value for the clock, the analyst will wish to select whether each object's clock is on (accumulating time against some DRM) or off (waiting for the occurrence of some future stimulus before beginning to accumulate time towards a DRM). The Clock On check box should be checked if the object's clock is to initially be set "on" and unchecked if the object's clock is to be initially set "off." This process is repeated for each object in the OBEST model following the same rules and conditions used in the Initial State drop-down menu.

The next task for the analyst is to define each object's state transition diagram. The analyst enters a special form for defining the state transition diagram from the Object Definition screen by pressing the State Transition Table button. The software responds to this command by bringing up the screen shown in Figure 4-4. Initially, this screen will be blank except for the list of defined model objects found in the upper left corner. To enter a state transition model for an object, the analyst first selects that object's name from the Object Name list. The software will respond by displaying a list of all defined states for this object in the States list, as well as a list of all defined events in the entire OBEST model. The analyst will choose from these lists to

define the state transition table. Initially, the Events list will be empty. The method for populating this list will be described shortly. Assuming, for a moment, that the Events list has already been populated with all of the events necessary to complete this object's state transition model, the analyst constructs the state transition model as follows. Note how, initially, the Current State box is enabled, and all other boxes related to this state transition are disabled. To enter a state transition, the analyst first selects the name of the "from" state (the state that this transition will remove this object from) from the States list, and presses the Add Current State button. The state's name will appear in the Current State box, and the focus will shift to the Next State box. The analyst now selects the name of the "to" state (the state that this transition will place the object into) from the States list, and presses the Add Next State button. Again, the state's name will appear in the Next State box, and the focus will shift to the Event box. The analyst now selects the name of the event that will cause this state transition to occur from the Events list, and presses the Add Event button. This event name will appear in the Event box, and the focus will shift to the Priority box. Here, the analyst will enter a number to indicate the priority of this state transition, as described in Chapter 3. The process is completed by pressing the Add Transition button, and can be repeated to enter additional state transitions for this object, or the analyst can select a new object from the Object Name list and repeat the process to enter state transitions for other objects.

State Transition Table

Object Name

Aircraft
Controller
Environment

States

Waiting
Taxi A
Taxi B
Taxi D
Taxi C
Taxi E

Events

Go D
Go E
Go F
Go G
Go H
Go I

Current State

Next State

Event

Priority

Add Current State

Add Next State

Add Event

State Transition Table

Current State	Next State	Event	Priority
Waiting	Taxi A	Go A	1
Taxi A	Taxi B	Go B	1
Taxi A	Taxi D	Go D	2
Taxi B	Taxi C	Go C	1
Taxi B	NoIncursion	Safe	2
Taxi D	Taxi F	Go F	1

Add Transition

Delete Transition

State Definition

Event Definition

State Transition Diagram

OK

Figure 4-4. Entering a State Transition Table.

The process of entering state transitions can be simplified through the use of shortcuts. The analyst can double-click on the state and events names, and they will be entered into the active boxes on the screen. Thus, the analyst first double clicks on the name of the “from” state, and that state’s name is automatically entered into the Current State box. Next, the analyst double clicks on the name of the “to” state, and that state’s name is automatically entered into the Next State box. The analyst double clicks on the transition event to enter that events name into the Event box. The analyst must still manually enter the numerical value for the state transition priority and press the Add Transition button to complete the definition process. Note that no two state transitions with the same “from” state should have the same numerical priority value. Recall that the purpose of this priority value is to break ties. That is, should the object be in this state and two different events happen simultaneously – each of which would cause a different state transition to occur in this object, the OBEST software will select the transition with the lower numerical priority. Thus, the first priority state transition should receive a numerical priority value of one, the second priority transition a numerical priority value of two, and so forth. As you can see, assigning more than one state transition the same priority value defeats the purpose of this important OBEST model feature.

Significant additional functionality exists on this screen. To delete an existing state transition, the analyst selects that transition from the State Transition Table at the bottom of the screen, and presses the Delete Transition button. Note that there is no “undo” operation for this state transition deletion command, so the analyst should use great care when pressing this button. The State Definition button returns the analyst to the form found in Figure 4-3. In this way, the analyst can enter names and definitions for any states that may not have been previously defined. The State Transition Diagram button is currently not operational, but eventually it is hoped that this button will lead the analyst to a graphical description of the state transition model as it currently exists on this form. The OK button returns the analyst to the Object Definition screen. Finally, the Event Definition button leads the analyst to the screen found in Figure 4-5. This screen provides a way for the analyst to populate the Events list in the upper right corner of the State Transition Table form.

The Event Definition screen is very similar to the State Definition screen shown in Figure 4-3, and functions in a very similar manner. The principal difference is that since states are specific to a particular object within the model, the State Definition screen required the analyst to select the object to which a new state would apply. Events, however, are not specific to a particular object, but are global to the entire OBEST model. For this reason, the Event Definition screen does not require the analyst to specify a state before entering event definitions. The list at the bottom of the form contains all of the events that currently exist in the OBEST model. The analyst adds new events to this list by typing an event name and event description in the appropriate boxes near the top of the form, and then pressing the Add Event button. Events can be deleted from the OBEST model if the analyst will highlight the name of the event to be deleted in that event list and press the Delete Event button. Note that there is no “undo” functionality for this delete command, so the analyst should exercise great care when pressing the Delete Event button (the Cancel button does not return the Events list to its original configuration). When all of the necessary events have been added to and deleted from the OBEST model, the analyst presses the OK button to return to the Event Definition screen.

Figure 4-5. Defining a Model Event

The next major phase of OBEST model development is to define the model's attributes and their values. From the Object Definition screen (Figure 4-2), the analyst presses the Attribute Definition button to bring up the Attribute Definition screen found in Figure 4-6. This screen provides a way for the analyst to view all of the attributes that have been defined for each object, and to set the initial conditions that will be applied to each attribute (i.e., define which of an attribute's values are to be used in that object's initial conditions). When one is initially developing an object model, this screen will be blank except for the list of objects found in the upper left corner. The analyst can define attributes that will be used in an object by pressing the Edit Attributes button (to bring up the screen shown in Figure 4-7), and can define discrete values that represent each attribute by pressing the Edit Attribute Values button (which brings up the screen shown in Figure 4-8). Once attributes and values have been entered into the model, the analyst selects an initial value for each attribute by selecting the object name and attribute name from the Objects and Attributes list, then selecting one of the defined attribute values from the Initial Value drop-down list to be used as the initial attribute value. There is no "undo" operation for the selection of an initial attribute value (the Cancel button does not return the list of attribute initial values to its original configuration). The OK button causes the software to return to the Object Definition screen, and should be used when the analyst is satisfied with the list of attributes and attribute values currently defined in the OBEST model.

The Edit Attributes screen is very similar to the State Definition screen (Figure 4-3), and functions in a very similar manner. The analyst selects the object to which the new attributes are to be applied from the Object Name list of the top of the form. The software responds by populating the attribute list at the bottom of the form with all of that object's existing attributes. The analyst enters a name and description for a new attribute, then presses the Add Attribute button to add this attribute to the list. This process can be repeated to add multiple attributes for this object, or a different object can be selected from the Object Name list to add attributes for

Figure 4-6. Summary of OBEST Model Attributes.

Figure 4-7. Entering Attribute Definitions.

Figure 4-8. Entering Values for Attributes.

other objects. If the analyst selects an attribute from the attribute list and presses the Delete Attribute button, that attribute is permanently deleted from the object model. There is no “undo” operation for the Delete Attribute operation (the Cancel button does not return the event list to its original configuration). The OK button causes the software to return to the Attribute Definition screen, and should be used when the analyst is satisfied with the attributes and initial values currently defined in the OBEST model.

The Edit Attribute Values screen (Figure 4-8) is shown when the Edit Attribute Values button is pressed on the Attribute Definition screen. This screen is conceptually similar to the Edit Attributes screen, except that attribute values are attributable to a particular attribute for a particular object, while attributes themselves are attributable only to a particular object. Therefore, to add or delete attribute values for a particular attribute, the analyst must first select the appropriate object from the Object Name list. This causes the software to populate the Attribute Name list with all attributes that have been defined for this object. When the analyst selects the attribute to be edited from this list, the software responds by populating the list of the bottom of the screen with all currently defined values for that attribute. The analyst can then add and delete attribute values using exactly the same steps used on the Edit Attributes screen (Figure 4-7) as described in the previous paragraph. The Value Logical check box is intended to restrict an attribute’s values such that it represents a Boolean quantity.

To this point we have focused on the capabilities within the OBEST software for

- definition of the objects included in the OBEST model,
- the definition of the states that will be used within each object to represent the different bodies of behavior that object is capable of assuming,
- the definition of the events that will be used within the OBEST model to transition objects between states,
- the state transition model (as implemented in a state transition table),
- the names and descriptions of the attributes that are used to describe each objects condition, and
- the various discrete values that each attribute is capable of assuming.

This describes all aspects of the object model except the IRMs and DRMs. As there is significant complexity involved in the definition of those response models, the entire section of this report has been dedicated to their development and editing.

4.3. Entering Immediate and Delayed Response Models

The construction of IRMs and DRMs in the OBEST software is exactly the same except for one minor detail: when entering a DRM, one must enter a value to represent the time delay that will elapse before that model's commands are executed. In the IRM, that delay time is, by definition, zero. In this section, we describe how an analyst can enter IRMs and DRMs into the OBEST software. But before we describe the mechanics of the software, we must establish some terminology. In this section, we discuss "cases" and "scripts." A case represents a logical statement that will be evaluated by the OBEST software as it seeks to establish which particular body of behavior is appropriate to apply under the existing set of model conditions. A script represents a series of commands that will be executed if the conditions are satisfied in the case to which it is attached. Recall that in Chapter 3 we discuss an "if – then – else if – then" methodology to embody IRM and DRM behavior. In our terminology, a case is what comes between the "if" and the "then," while a script is what comes after the "then."

As one might expect based on the previous section, every case and every script has a name and a description. Both cases and scripts can be used multiple times within the OBEST model. Why would one want to do this? One might want to use a script several times if the same body of behavior (the same sequence of commands) might be used by an object under a variety of conditions or in different states. A script might also be reused for different probabilistic branches within the same DRM model when there is some probabilistic distribution to represent the delay that might occur before that script's events come to pass. Note that, since an object can only change its own attributes, and since the commands in a script can change an object's attributes, each script can only be executed by the object for which it was created. Any object, on the other hand, can use cases, because they simply read the attributes of all objects in order to make a logical decision as to whether a script should be applied under a particular set of

circumstances. Cases can be reused within the same object if one wishes to respond to the same set of circumstances while the object is in different states. They can be reused within different objects if more than one object needs to respond to the same set of circumstances.

By breaking the OBEST IRM and DRM models into cases and scripts, an interesting phenomenon occurs: one can mix and match scripts and cases in a variety of ways to achieve whatever behavior is desired. Thus, if the same body of behavior can be induced by different circumstances when an object is in different states, one can combine different cases with the same script to achieve that behavior. If the same circumstances produce different behaviors when object is in different states, the analyst can combine the same case with different scripts to produce that effect. When probabilistic branching is used, one takes a single case and probabilistically associates it with more than one script, more than one delay, or some combination thereof. The use of named cases and named scripts makes it very easy to reuse these model elements, and in so doing, efficiently create an OBEST model while at the same time making the model quality assurance task somewhat easier.

Let us now describe the construction of an OBEST IRM. From the Object Definition screen (Figure 4-2), one presses the Immediate Response Model button to bring up the IRM screen shown in Figure 4-9. When one begins construction of a new OBEST model, this screen is blank except for the two lists on the upper half of the form: the Object Name list and State Name list. The IRM is specific to each state for each object, so these lists enable the analyst to select a combination of object and state for IRM development in the same way that a combination of object and attribute were selected in the Edit Attribute Values screen (shown in Figure 4-8). Once one selects the object and state, the details of the IRM (as it currently exists) are shown at the bottom of the screen. The cases that have been assembled to make up that model are shown in the IRM Cases list. Each case can be associated with more than one script if probabilistic branching is used. Thus, when the analyst selects a case from the IRM Cases list, the software responds by showing the scripts associated with that case (and their associated probabilities) in the Scripts list. In Figure 4-9, the Aircraft object's Waiting state has a complex IRM consisting of many cases. Here, the C-Dest-OK-LowHP case is associated with three scripts: SetDest-25R will be executed with a probability of 0.995, SetDest-25L will be executed with a probability of 0.003, and SetDest-24R will be executed with a probability of 0.002. Selecting other objects, states, or cases will show different tables of scripts and probabilities.*

When one begins construction of a new OBEST model, the bottom half of this screen is blank. If one were to press the Case Editor or Script Editor buttons, one would find those screens to be similarly blank. In order to facilitate the discussion of IRM model development, we will (for the moment) assume that one has already entered appropriate cases and scripts, and that one is attempting to combine those existing model fragments to form a complete IRM. The screens used to develop cases and scripts will be discussed shortly. To construct an IRM from scratch, one first presses the Add Case button to reveal a simple dialog that lists all available case names.

* The only difference between this screen and the corresponding DRM screen is that the Scripts table on the DRM screen has three columns: Script, Probability and Delay.

When the analyst selects a case and presses the OK button, that case name is added to the IRM Cases list. One can add many cases at once, or add the cases individually and populate the associated scripts at the same time.

The screenshot shows the IRM dialog box with the following components:

- Object Name:** Aircraft, Controller, Environment
- State Name:** Waiting, Taxi A, Taxi B, Taxi D, Taxi C, Taxi E
- Buttons:** Case Editor, Script Editor, Add Case, Delete Case, Add Script, Edit Script, Delete Script
- IRM Cases:** C-Dest-24R, C-Dest-OK-LowHP (selected), C-Dest-OK-MedHP, C-Dest-OK-HiHP, C-Rt-Right, C-Rt-OK-LowHP, C-Rt-OK-MedHP, C-Rt-OK-HiHP
- Scripts Table:**

Script	Probability
SetDest-25R	0.995
SetDest-25L	0.003
SetDest-24R	0.002
- OK** button at the bottom.

Figure 4-9. Entering an Immediate Response Model.

To associate scripts with a case, one first selects the appropriate case name to reveal any existing associated scripts. When the analyst presses the Add Script button, the software responds by generating a very simple dialog that lists all available script names. The analyst selects the appropriate script name and enters the associated probability on this form, then presses the OK button to add this script/probability combination to the Scripts list. The process is repeated until all scripts associated with this case are selected.* To add scripts for a different object, state, or case, simply select the appropriate names and repeat this process. Once all appropriate IRM cases and scripts have been added and/or deleted, the analyst presses the OK button to return to the Object Definition screen.

The Delete Script and Delete Case buttons on this form operate in the same manner as the delete buttons on other forms in this software package. To delete a script from an IRM model, select it

* The sum of the probabilities in the Scripts list should equal one once the last script is added. However, the software does not currently check to ensure that this criterion is satisfied.

in the Scripts list and press the Delete Script button. Similarly, to delete a case, select the appropriate case in the IRM Cases list and press the Delete Case button. Note that when a case is deleted, all of its associated scripts are also “deleted” in the sense that they are no longer associated with that IRM. To recreate this IRM, one would have to first add the case and then add any necessary scripts using the methods described above. There is no “undo” capability for either of these delete commands, so the user is encouraged to use great care when pressing these delete buttons.

The astute reader may remember that the functioning of the “if – then – else if” IRM and DRM methodology was such that the software executes the script(s) for the *first* case that evaluates as “True” and *ignores* all subsequent cases – regardless of whether they would evaluate as “True” or “False.” Thus, the order of the cases in an IRM or a DRM is critically important. The analyst can change the order of the cases by using the arrow buttons to the left of the IRM Cases list. Simply stated, the selected (highlighted) case name is moved up or down one position in the IRM Cases list for every time the Up Arrow or Down Arrow button is pressed, respectively. This gives the analyst total control to order the cases in any way that would produce appropriate IRM behavior.

Figure 4-10. Entering a Delayed Response Model.

The screen used to construct a DRM is shown in Figure 4-10. From the Object Definition screen (Figure 4-2), one presses the Delayed Response Model button to bring up the DRM screen. This screen is identical to the IRM screen in all respects except that it enables the analyst to enter and view the delay times that must be associated with DRM scripts. The Add Case, Add Script,

Delete Case, Delete Script, and OK buttons all behave as described previously for the IRM screen. Note that the same cases and scripts that were developed for the IRM can also be used in the DRM if it would be appropriate to do so. There is no need to enter separate cases or scripts for use in the DRM.

The foregoing discussion showed how an analyst could create IRMs and DRMs under the assumption that cases and scripts had already been entered. Of course, when one is initially developing the OBEST model, this assumption will not hold because the list of cases and the list of scripts will both be empty. To enter and edit cases, the analyst presses the Case Editor button on the IRM or DRM screen. This action causes the software to bring up the screen shown in Figure 4-11. Initially, this screen will be blank because cases are not associated with individual objects.

Object Name	Attributes	Attribute Values	Value Logical
Aircraft	Traffic	Clear	False

Figure 4-11. Entering Logic Cases for Immediate and Delayed Response Models.

The first step in defining a new case is to add its name and description to the cases list in the upper left corner of the screen. To do this, the analyst presses the And New Case button and enters a name and description for the new case in the dialog provided. This sets up the structure for the case that will be populated with logical statements. To enter those statements for a new case, the analyst first selects the case's name from the Cases list. The existing logic for this case, if any, will be described in the Objects, Attributes, and Values lists at the top of the screen. Since this is a new case, these lists will be empty. The logical statement that makes up this case will be composed of attribute values – possibly from several different objects – that must take on specified values in order for the logical statement to evaluate as true.

Before we describe how those attributes and values are incorporated into a case, it is important to understand how the OBEST software will ultimately evaluate this logical statement. Simply stated, if more than one value is specified for a particular object/attribute combination, the OBEST software will use a logical OR condition between them. In other words, for a “bicycle” object and “color” attribute, specifying the values “red,” “blue,” and “yellow” would cause the software to respond positively if the bicycle were red, yellow, or blue, and negatively for all other values. On the other hand, when the logical statement contains values from more than one attribute (or even more than one object), the OBEST software will use a logical AND condition between them. Consider a “weather” object with attributes “sky conditions” and “temperature.” A case that specifies a value of “cloudy” for sky conditions and a value of “cold” for temperature would only respond positively to weather conditions that are both cloudy and cold. Conditions that are cold but not cloudy, or cloudy but not cold, or neither cloudy nor cold would not elicit a positive response from the OBEST software. Thus, an OBEST case represents a logical AND between objects and attributes, combined with a logical OR between values within a single object/attribute combination.* Should the analyst wish to develop logical conditions that are more complicated than those enabled by this method, the analyst must break those conditions into multiple cases that can be evaluated sequentially, each one of which must follow these rules.

We have just described how each logical case is composed of a number of conditions and that are evaluated according to a predefined set of AND and OR logical rules. Each condition is based on a single object, attribute, and attribute value combination. These conditions can be entered into the case in any order. The controls that are used to enter a single condition into the case are located in the Conditions frame near the bottom of the screen. Once the analyst has selected the case to be edited from the Cases list, the analyst selects the object that contains the attribute and attribute value that are eventually to be used by clicking on the appropriate entry in the Object Name list. The software responds by displaying all defined attributes for that object in the Attributes list. When the analyst selects the appropriate attribute, the software displays all of the defined attribute values for that object/attribute combination in the Attribute Values list. The analyst then selects whether this object/attribute/value combination is to be true or false using the Value Logical check box, then presses the Add Condition button to enter that condition into the case. The process is repeated for other conditions (object/attribute/value combinations) until the case definition has been completely entered. The analyst can check the conditions that have been entered into the case using the Objects, Attributes, and Values lists at the top of the screen, as described previously. To add additional conditions to a previously defined case, the analyst selects that case from the Cases list and follows this same process to enter additional conditions. Additional cases can be defined by pressing the Add New Cases button and then

* Note that the OBEST methodology also permits the analyst to specify the logical value of the attribute value being used. Typically, this value is set to “True” as used in the previous examples. If this value were set to “False,” the weather example would elicit a positive response from the OBEST software under conditions that were neither cloudy nor cold. This is because the OBEST software would be evaluating the logical statement “NOT cloudy AND NOT cold” in the object model. Similarly, in the bicycle example, the software would respond positively to bicycles that were neither red nor yellow nor blue because the OBEST software would be evaluating the logical statement “NOT (red OR yellow OR blue).”

repeating this process. The analyst can define the names of several new cases first, and then populate the conditions of each later, if desired.

The original plans for this software would have enabled the analyst to do a number of editing operations on a case using the buttons in the lower left-hand corner of this screen. These functions, however, have not been fully implemented at this time. Currently, the accepted method for dealing with errors in a case that cannot be corrected with additional conditions is to delete the case and re-enter the information. A case is deleted when the analyst selects the case from the Cases list and presses the Delete Case button. There is no “undo” capability for the Delete Case operation (the Cancel button does not return the cases list or the case definitions to their original configuration). The OK button causes the software to return to the screen that invoked the case editor (either the IRM or DRM screen). It should be used when the analyst is satisfied with the set of cases that have been defined in the OBEST model.

The remaining task in developing an OBEST model is to specify the scripts that will be executed as part of the IRM and DRM. To enter and edit scripts, the operator presses the Script Editor button on the IRM or DRM screen. This causes the software to display the screen shown in Figure 4-12. Initially, the Script Name list and the Script Description box on this screen will be blank if one is developing an OBEST model from scratch.

The screenshot shows a window titled "Script Editor" with a standard Windows-style title bar. The window is divided into several sections:

- Object Name:** A list box containing "Aircraft", "Controller" (selected), and "Environment".
- Script Name:** A list box containing "GoTo-25R", "NoHoldShort" (selected), "GoTo-Right", "GoTo-25L", "GoTo-24R", and "DoNothing".
- Script Description:** A text box containing "Correct destination w/o hold short in".
- Add New Script:** A button located to the right of the Script Description box.
- Attribute Events:** A section with three sub-sections:
 - Attributes:** A list box containing "Destination", "Route" (selected), "HoldShort", and "Stop".
 - Attribute Values:** A list box containing "Unset", "TaxiLeft" (selected), and "TaxiRight".
 - Script Attributes:** A text box containing "Destination = 25R", "Route = TaxiLeft", and "HoldShort = Go".
- Buttons:** At the bottom of the Attribute Events section are "Add", "Attribute Definition", and "Delete" buttons. At the very bottom of the window are "OK" and "Cancel" buttons.

Figure 4-12. Entering Script Definitions for Immediate and Delayed Response Models.

Each script is tied to a particular object because scripts can change the attributes of an object, and an object is only allowed to change its own attributes. Thus, the analyst must select an object from the Object Name list before that object's scripts will be listed in the Script Name list. To enter a new script, the analyst selects the name of the object to which it will apply, then presses the Add New Script button. The software provides a dialog that will enable the analyst to enter a name and description for this new script. When the analyst completes the form, the software enters that new script name into the Script Name list. Consistent with previous screens in the OBEST software, the analyst can enter names and descriptions for the scripts one at a time (defining each script as it is named), or the analyst may enter a large block of names all at once and then define each script later at his or her convenience.

When the analyst selects an object from the Object Name list, the software responds by displaying the list of all script names that have been defined for that object in the Script Name list. When a name is selected from this list, the software displays the current definition for that script in the lower half of the screen. Each script can have three different types of elements: Attribute Events (commands that will cause this object to change the value of its attributes), object model events (entered on the Events tab, which may cause this object or other objects to change state), and system events (entered on the Special Events tab, which may cause termination or truncation of the scenario or cause this object to manipulate the status of its internal clock).^{*} Each type of event, as well as the facility to add or delete events of that type, is found on the appropriately named tab. As each tab is displayed, the right hand section of the screen displays those events that are already part of the script definition, while the left hand section of the screen displays those events that are available for the analyst to add to the script definition. Each tab also has a button for deleting particular events from the script definition. One deletes events from the script definition in the same manner as one has deleted other parts of the object model on other screens: one highlights the events to be deleted and presses the Delete button. There is no "undo" capability for either the Add or Delete functions on the Script Editor screen (and the Cancel button does not restore the script definitions to their original configuration), so the analyst must be careful when adding and deleting items from scripts.

To add an event to a defined script, the analyst begins by selecting the object and script to which this event will apply from the Object Name and Script Name lists. The analyst then selects the tab that is appropriate to the type of event to be added to the script from the bottom of the screen. The Events and Special Events tabs contain simple lists of events that can be added to the script. The analyst simply highlights the desired event and presses the Add button to include this event in the script. Attribute Events are defined in a similar manner, although the analyst must define both the attribute to be modified (from the Attribute list) and the value to which it is to be set (from the Attribute Values list) before pressing the Add button to include the event in the script.[†]

^{*} Only one Special Event is allowed per script.

[†] Recall that, by convention, an object can only modify its own attributes. Thus, when defining Attribute Events, it is not necessary to state which object's attributes we want to modify. This also explains why the script cannot be defined outside the context of an object, while a case is not so limited.

Note that the script definition at the right of the screen lists both the attribute and its value as being part of the script.

4.4. *Evaluating an OBEST Model*

By following the steps in the preceding sections, we have now completed the definition of an OBEST model. Currently there are very few automated checks in the software to determine whether a completed OBEST model is fully specified and self consistent. Indeed, it is very difficult for an automated facility to make such checks because it requires one to determine whether particular features of the model were included simply for completeness or later elaboration (in which case their lack of use does not represent an error), or they are critical components of the object model (in which case the fact that they are not used will cause the object model to behave very differently from the analyst's expectations). It is possible that future enhancements to the software will allow more of these features to be added.

One of the fundamental advantages of an object-based model is that, after the analyst encapsulates the behavior of the system in objects, one can simulate a wide variety of scenarios and conditions by simply changing the initial conditions or boundary conditions imposed upon the object model. In OBEST, this is accomplished by presetting particular attributes to specific values that might define a desired scenario, and by "seeding" the analysis with model events that are to occur at the beginning of the scenario. To impose these initial conditions on the system, one begins at the main OBEST screen (Figure 4-1) and presses the Initial Conditions button. When this occurs, the software responds by displaying the Initial Conditions screen shown in Figure 4-13. Initially, this screen provides a facility for setting initial condition attribute values for the OBEST analysis. The analyst sets these conditions by selecting the object, attribute name, and attribute value to be imposed on the system from the Objects, Attributes, and Values lists on the left side of the screen. When the desired combination has been selected, the right arrow button causes it to be entered as an initial condition in the Set Attributes list. To delete a previously entered attribute, one highlights the combination in the Set Attributes list and presses the left arrow button.

Figure 4-14 shows the screen used to seed the analysis with model events that are to occur at the beginning of the scenario. These events are entered using exactly the same steps described for Figure 4-13: an event is selected from the Available Events list and the right arrow button is pressed in order to enter that event into the Initial Events list, while an event is deleted from the Initial Events list when its name is highlighted and the left arrow button is pressed (note that there is no "undo" capability for any of the actions related to the selection of initial conditions).

Another important part of the preparation to execute an OBEST model is to specify any termination conditions that one wishes to impose on the system. The system automatically terminates a scenario when it reaches "equilibrium conditions" – that is, when all IRMs are in equilibrium and no further DRMs are active. However, the analyst may wish to terminate the scenario under more restrictive conditions in order to save computational time or because further analysis is simply beyond the scope of the problem to be solved. To impose additional termination conditions, the analyst proceeds to the main OBEST screen (Figure 4-1) and presses

Figure 4-13. Setting Alternate Initial Conditions for the Model.

Figure 4-14. Setting Events to Initialize the OBEST Model.

the Termination Conditions button. This causes the software to bring up the screen shown in Figure 4-15. While not all of the options displayed on this screen are fully functional, it is instructive to consider the variety of methods one might use to terminate a scenario.*

Figure 4-15. Setting Path Termination Conditions.

The simplest termination condition is time. The analyst may only be interested, for example, in events that occur within 24 hours of scenario initiation. Such a termination condition is implemented using the Time tab on the Termination Conditions screen. A second, simple termination condition can be based on an attribute value. The analyst may have as a principal purpose for an analysis the simple determination of whether or not a particular attribute value is realized. For example, on a telephone central office object, the analyst may be most interested in

* It is also possible to terminate scenarios by embedding Special Events in object scripts. One such event, Terminate Scenario, causes the system to immediately terminate the scenario branch being processed and include it in the final results tabulation. Another such event, Truncate Scenario, also causes the system to immediately terminate the scenario branch being processed. However, truncated scenarios are not included in the final results tabulation. The analyst uses a Truncate Scenario command when the OBEST model reaches a condition that indicates that this scenario is not of interest to the analyst's purpose in performing this OBEST analysis. One might also wish to truncate a scenario because of its extraordinarily low probability in order to save the computational time that might be incurred should further branching of the scenario occur, but this facility is not currently implemented in the OBEST software.

whether the battery charge attribute reaches a value of “dead” before equilibrium conditions are established when commercial power is restored. The Attribute Values tab provides a facility whereby the analyst can select one or more attribute values as termination conditions. A scenario will be terminated should any one of these attribute values be realized at any time during a scenario. A similar set of termination conditions can be imposed when an object enters a state that is of particular interest to the analyst. For example, if one state of a telephone central office object were “operating under commercial power,” the analyst might specify that entry into this state provides an adequate termination condition for scenarios that are initiated with a loss of commercial power (blackout). The Object States tab enables the analyst to select one or more object model states for use as termination conditions.

The final set of termination conditions envisioned for the OBEST software can be thought of as a direct parallel to the concept of cases that was developed for the IRMs and DRMs. The analyst might put together a set of logical conditions (a combination of object model attributes) which, should it become true at any point during the analysis, would indicate that it is appropriate to terminate the scenario being evaluated. This functionality will ultimately be provided under the Logical Conditions tab on the Termination Conditions screen.

The final task that the analyst needs to accomplish before actually running the OBEST model is to determine which features of each scenario will be saved for presentation as results to the analyst. This process, known as “binning,” was initially described in Section 3.3.3.* The OBEST software does not allow the analyst to save every scenario generated through evaluation of the object model because an object model of even moderate complexity will likely generate thousands of probabilistic scenarios. Particularly complex object models may generate millions of scenarios. It is unreasonable to expect a human analyst to wade through thousands or millions of scenarios and extract meaningful insights. Also, given the time-dependent behaviors that occur within these scenarios, it is unreasonable to expect a desktop computer to save all information for thousands or millions of scenarios. Memory and disk storage space are limited, and the sheer computational effort involved in just the input/output operations to write such information to disk would become extremely burdensome.

Fortunately, the analyst is usually interested in only a subset of the information that is actually generated as a scenario is solved. Our objective, then, is to save all information that is relevant to the analyst’s purpose in performing the analysis, while throwing away all irrelevant information. Furthermore, we combine those scenarios that are identical in all aspects that are important to the analyst into a single “bin” that acts as a surrogate for all of these similar scenarios. However, if the software is to save only relevant information, the analyst must define for the software what “relevant” means. This is done on the Bin Categories screen shown in Figure 4-16. The analyst reaches this screen from the main OBEST software screen (Figure 4-1)

* By way of review, one can think of specifying bin definition information according to “topics” on which information will be saved (known as dimensions), and “categories” within each topic that represent the discrete values to be saved (known as values). For example, a topic or dimension might be the color of an object, and the legal categories or values for the color might be red, yellow, green, blue, black, and white.

by pressing the Bin Categories button. Using the five tabs on this screen, the analyst can cause the software to save answers to the following questions:

- What value did this attribute take on at the end of the analysis? (Answered on the Attribute Values tab)
- What state was this object in at the end of the analysis? (Answered on the Object States tab)
- Within which time window did this logical condition become true? (Answered on the Time Conditions tab)
- In which order did these logical conditions occur? (Answered on the Order Conditions tab)
- Which one of these all-inclusive logical conditions was true at the end of the analysis? (Answered on the Logical Conditions tab)

Figure 4-16. Binning Model Results Based on Attribute Values.

The Attribute Values and Object States tabs work in a manner similar to the Initial Conditions screen. On the Attribute Values screen, the analyst selects the name of the object and attribute for which values are to be saved from the Objects and Attributes lists, then presses the arrow button to add that attribute name to the Dimensions list. Similarly, on the Object States tab, the analyst selects the name of the object for which the final state is to be saved and presses the

arrow button to add that name to the Dimensions list. A dimension is removed from this list when the analyst selects the dimension's name and presses the appropriate arrow button.

The remaining three tabs on the Bin Categories screen all make use of logical conditions that are similar to the cases described in the previous section. The analyst can define these logical conditions using an editor that is accessed by pressing the Bin Condition Editor button. Since the bin condition editor is relatively similar to the case editor described previously, it will not be described here. Once an analyst has defined these logical bin conditions (which, as you might expect, are identified by name and description fields), these conditions can be used to specify time, order, or logical conditions. When specifying a time condition, the analyst will select one logical bin condition as well as a series of breakpoints between time intervals. The OBEST software will determine whether this logical condition occurs at all, and if so, the time interval during which it first occurs. When specifying order conditions, the analyst will select two or three logical bin conditions, and the OBEST software will determine the order in which these conditions occur (if they occur at all). Finally, when specifying logical conditions, the analyst will select two or more logical bin conditions that they believe to be fully inclusive (i.e., every scenario will satisfy one of these logical conditions). The OBEST software will determine which of these conditions is true at the end of the scenario.

When the analyst has defined all necessary binning conditions, they return to the main OBEST program screen by pressing the OK button. The OBEST model will now be analyzed if the analyst presses the Run Scenario button. While the software is processing scenarios, it keeps the analyst apprised of its progress through the various indications shown on the model progress screen (Figure 4-17). This screen provides the analyst with feedback regarding:

- How long processing has been running,
- The current state of the solution process (the number of entries in the scenario stack [see Section 3.3] and the probability of the scenario currently being assessed),
- The number of scenarios that have been processed so far (broken down into scenarios that have been binned and scenarios that have been truncated), and
- The total probability of all scenarios that have been processed so far (again, broken down into scenarios that have been binned and scenarios that have been truncated).

If all of the IRM and DRM models in the OBEST model have been developed such that their probabilities all sum to one, then the analyst can expect the scenario processing to be completed when the sum of the Cumulative Probability and Truncated Probability reaches one. This can be somewhat deceiving, however, for two reasons: first, numerical round off occurs as one accumulates the sum of a long list of numbers (thus, the sum may not reach exactly one or may even slightly exceed one when the analysis is complete), and second, the numerical algorithm implemented in the OBEST software works to discover the most likely scenarios first. Thus, the sum will increase quickly at first, then may dramatically slow down during the time when the final scenarios are being discovered. To get an idea of how fast this sum is changing, the analyst should examine the Scenario Probability box, as this indicates the most recent value that was added to this sum.

Figure 4-17. OBEST Software Display During Model Analysis.

This screen provides two additional capabilities to the analyst. First, the analyst may wish to terminate evaluation of the OBEST model before the solution process is complete. The analyst accomplishes this by pressing the Halt Scenario Evaluations button. Note that the software cannot resume the solution process where it left off, so any reanalysis of this scenario must start at the very beginning. Second, many analysts realize that the frequent update of a display can divert significant computational resources away from the analysis task itself. The OBEST software enables an analyst to maximize the resources devoted to model solution by providing a Turn Off Progress Displays button which, when pressed, causes the progress displays to freeze at their current value even though the model solution process is continuing. The label for this button changes to Turn On Progress Displays (as seen in Figure 4-17) during the time that progress displays are frozen, and the analyst simply presses this button a second time to restore the progress displays to their original form.

The OBEST software also provides the analyst with a powerful software debugging tool. To use this tool, the analyst starts from the main OBEST software screen (Figure 4-1) *before* pressing the Run Scenario button, and selects the Scenario Debug Form entry from the View menu. This action opens the Scenario Debug window shown in Figure 4-18 as a second window that parallels the model progress screen (Figure 4-17). Once the Scenario Debug window is open, the analyst presses the Run Scenario button as usual to begin solution of the OBEST model. With the Scenario Debug window open, the analyst can cause the solution algorithm to pause after each model generation, IRM execution, or DRM execution. The next step is taken when the

Continue button is pressed. Using the tabs at the bottom of the screen, the analyst can examine virtually any part of the OBEST solution process.

Figure 4-18. OBEST Software Debugging Screen.

Once the OBEST software completes its analysis of this object model, the analyst will wish to view the binned results. To do so, the analyst selects the Bin Results entry from the View menu to reveal the screen shown in Figure 4-19. Initially the screen is blank except for the entries in the Unused bin dimensions list. The analyst selects the first dimension that is to appear in the bin results table from this list and presses the left arrow button to add it to the Used bin dimensions list. The software responds by placing those values that were found for this bin dimension in the table along with the probabilities that were realized for those values. Should a particular value for a bin dimension be theoretically possible yet never be actually observed during the analysis, it is omitted from the table. The analyst repeats the bin dimension selection process to add additional columns to the table. In Figure 4-19, three dimensions have been added to the table (as seen in the Used dimensions column) and one remains unused. Note that the table contains all combinations of values that were observed for these attributes. Thus, the third line in the table represents the situation where the value for the Destination dimension was found to be “25R”, the value for the Route dimension was found to be “TaxiRight”, and the value for the HoldShort dimension was found to be “Hold.” Scenarios where even one of these dimension values was found to be different are found on different lines in this table. The analyst can examine different combinations of dimensions by deleting dimensions from the Used dimensions column (using the right arrow button) and/or using additional dimensions as

described above. The table is automatically updated, enabling the analyst to rapidly slice the model results and distill global insights.

The screenshot shows a window titled "Bin Results" with a table of results and two filter sections below it. The table has four columns: Destination, Route, HoldShort, and Probability. The filter sections are labeled "Used" and "Unused". The "Used" section has a list box with "Destination", "Route", and "HoldShort" (selected). The "Unused" section has a list box with "Aircraft". There are also two arrow buttons between the filter sections and an "OK" button at the bottom.

Destination	Route	HoldShort	Probability
25R	TaxiLeft	Hold	0.994866027189522
25R	TaxiLeft	Go	1.00123916570015E-04
25R	TaxiRight	Hold	9.96988699211023E-04
25L	TaxiLeft	Hold	5.00517493350375E-05
24R	TaxiLeft	Hold	0.000724570608433

Used

Destination
Route
HoldShort

Unused

Aircraft

OK

Figure 4-19. Displaying the Results of an OBEST Analysis.

4.5. Summary

In this chapter we have provided a description of the software that was developed to implement the OBEST modeling methodology. Because of the budgetary limitations involved in this LDRD project, the software has some rough edges, and a variety of features and enhancements remain to be developed. The software should be viewed as demonstration software rather than production software for these reasons. However, even in its current state, the software does provide a powerful tool for developing and analyzing OBEST models. It is capable of discovering the thousands or millions of possible scenarios that exists in a typical OBEST model and categorizing them into binned results. It is further capable of helping the analyst slice those binned results to develop genuine insights into the behavior of the model and, it is hoped, the behavior of the system that this model is designed to represent. As such, it provides functionality to implement a critical piece of the overall OBEST modeling methodology.

This Page Intentionally Blank

5. Example Problems

As part of this LDRD project several example problems were constructed to demonstrate the application and explore the capabilities of the OBEST methodology. These example problems range from modeling a racquetball game to aircraft runway incursions. The OBEST system model for each example problem represents the functionality and behavior of the system. The model is constructed by identifying the system's blocks of functionality, and documenting the interactions, behavior and attributes of those functional blocks.

The examples in this section closely follow the methodology described in Chapter 1. By way of review, the OBEST system model for each example problem is developed using three views: a system structure diagram, an interaction diagram, and a state transition diagram. The *system structure diagram* presents the major blocks of functionality and specifies how these blocks relate to one another. A second view, the *interaction diagram*, captures the dynamics of the system at “black box” levels, and identifies the “flows,” or functional interactions that occur among the various blocks in the system structure diagram.

The different behaviors that a functional block can exhibit and the ways in which these behaviors change are documented in the third view, a *state transition diagram*. Each type of behavior is represented by a *state*. State transitions indicate the ways in which the behavior of a functional block can change, and are caused by *events*. In a system model, each functional block is an object with a state transition diagram that may include one or several different states. The transitions specify the state to which an object moves if the required event occurs. Any state can be associated with multiple transitions. Events may be generated internally to the object when a specified state, set of attribute conditions, or time is reached, or may be externally generated when conditions specified by another object are reached or by the analyst.

The details of the object's behavior while it is in each state (how the object responds to stimuli and the way its attributes change) are specified in the state transition diagram using sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models. Recall that each “script” is a series of actions that will be taken by the object, and may include the generation of object model events (to cause this or another object to change states), changing the values of its own attributes,^{*} and/or generation of special “system events.”[†] Depending on the domain and complexity of the problem, the sets of “if – then – else if – else” scripts and logical conditions for each state can be quite lengthy.

^{*} An attribute represents a changeable, measurable and calculable characteristic of the object, and is restricted to discrete and Boolean values. To simplify modeling considerations, each object is only allowed to modify the values of its *own* attributes, and cannot directly modify the values of attributes in other objects. The IRM and DRM models for those objects must be “aware” of the objects around them and change their attribute values accordingly.

[†] System events relate to the management of the object model, and may include such events as starting, stopping or resetting delay clocks, or terminating or truncating the evaluation of a scenario.

The example problems were constructed concurrently with the development of the OBEST demonstration software. For each example problem, the system model was constructed. Subsequently, the system models for selected example problems were implemented in and analyses were performed using the OBEST demonstration software. This section describes six example problems developed during the project, including the following:

- Electric power supply for a police station;
- Fuel tank supply for a gas burner;
- A singles racquetball game;
- Circuit analysis for nuclear power plant cable fires;
- A homeowner's "Y2K" preparation; and
- Aircraft runway incursions.

Several of these examples relate to infrastructure outages and how they might affect particular systems. For each example problem, a description of the system model is provided. This description includes the graphical representations and narrative text for the system structure diagram, the interaction diagram, and the state transition diagrams. For the state transition diagrams, a narrative summary of the "if – then – else if – else" scripts and logical conditions is provided. Complete description of each example problem including the "if – then – else if – else" scripts are provided in various appendices to this report. Additionally, for the selected examples, implementation of the models with and analysis results from the demonstration software are discussed.

5.1. Electric Power Supply for a Police Station

The first example problem constructed for this project considered electric power supply for a police station. This example problem was constructed to demonstrate some of the basic ideas and capabilities of the OBEST methodology for modeling infrastructure systems, specifically consideration of variable event ordering and interrelated infrastructure elements to determine probabilistically-weighted scenarios. Note that the information in this sample problem is completely fictional. The problem is vaguely tied to reality in that a very large-scale electric power outage (on the scale envisioned by some for the "Y2K" problem) would be difficult to restore without functional telecommunications links to coordinate that effort. Beyond that, any connection with a "real" scenario is tenuous at best. However, the problem is simple enough to describe and model (and related to the original infrastructure-related thrust of this LDRD project) that it made a good first example problem for the project.

5.1.1. Problem Description

In this problem, the commercial power supply for a police station and a telephone central office switch building is provided from a single electric power bus operated by the electric power company.* If the power bus fails, the speed with which the power can be restored depends on whether telephone service remains operational. If power fails but telephone service continues to operate, it is 90% likely that commercial power can be restored within 2 hours. Without telephone service, it takes at least 36 hours to restore electric power. It also takes at least 36 hours to restore power with operating telephone service if power is not restored within 2 hours as this represents a major failure within the electric power distribution network.

The telephone central office building has generator backup power for 48 hours, and that generator starts and runs properly 97% of the times it is demanded. The building also has battery backup power for 8 hours, and for the sake of simplicity, that power supply is assumed to be completely reliable. The same telephone central office switch provides telephone service for a police station. The police station has generator backup power for 24 hours, and that generator starts and runs properly 95% of the times it is demanded. The police station has a 90% likelihood of success in obtaining additional fuel for their generator (i.e., maintaining power for more than 24 hours) if they do so before the telephone switch loses power (8 hrs, if the telephone central office generator does not start). The initiating event for this example problem is that the electric power bus fails. Of concern are the event scenarios that lead to power failure for the police station up to 36 hours after the electric power bus fails.

5.1.2. Object Description

The system structure diagram for this example problem is shown in Figure 5-1. The functional blocks are defined as the telephone central office switch building (TSB), police station power (PSP), and the electric power bus (EPB). The lines between the functional blocks indicate that interactions exist between the telephone switch building and the police station power, the telephone switch building and the electric power bus, and the electric power bus and the police station power. The interaction diagram is provided in Figure 5-2. The functional interactions between the blocks are indicated by directed arrows and are specified in Table 5-1.

This section presents descriptions for the objects that represent the three functional blocks shown in Figure 5-1. Each description includes a state transition diagram (with relevant events listed) and a narrative of the “if – then – else if – else” scripts and logical conditions for the IRM and DRM for each state. The detailed “if – then – else if – else” scripts and logical conditions are provided with the complete model description in Appendix A.

* Here a “bus” refers to a main interconnection point within the power company’s distribution system that supplies power to a particular local area.

Figure 5-1. System structure diagram for police station power model.

Figure 5-2. Interaction diagram for police station power model.

Table 5-1. Functional interactions for police station power model.

TSB-EPB:	The telephone switch building provides telephone service to support the electric power bus.
TSB-PS:	The telephone switch building provides telephone service to the police station.
EPB-TSB:	The electric power bus provides normal power to the telephone switch building.
EPB-PS:	The electric power bus provides normal power to the police station.

The state transition diagram for the electric power bus, as shown in Figure 5-3, includes two states and two events. Within this model, the electric power bus object provides power for the police station and the telephone switch building. The two states for the object, then, are (Provides Power) and (Does Not Provide Power). The events that cause transition between these two states are *Power fails*, and *Power restored*. The electric power bus object has one attribute that reflects whether the object provides or does not provide power (this attribute is set to the appropriate value in each state's IRM). Recall the initiating event for this problem is that the electric power bus fails. When this occurs, normal power is no longer available for the telephone switch building or the police station. Again, restoring normal power is dependent on the telephone service remaining operational.

Figure 5-3. State transition diagram for the electric power bus.

The State:(Provides Power) has no DRM and only the IRM described above. The State:(Does Not Provide Power) also has the IRM described above. The DRM for this state generates the Event:*Power restored* at the appropriate time to reflect the time at which commercial power is restored. It has two cases: First, if telephone service is available (determined by examining an attribute of the telephone central office switch), the event is generated after a 2 hour delay (90% probability) or after a 36 hour delay (10% probability). If telephone service is not available, the event is only generated after a 36 hour delay.

The state transition diagram for the telephone central office switch building is shown in Figure 5-4. Within this model, the telephone central office switch object provides telephone service for the police station and also to support the electric power bus. This object includes four states, (Available With Normal Power), (Available With Generator Power), (Available With Battery Power), and (Unavailable). The telephone central office switch object has one attribute that reflects whether the object provides or does not provide telephone service, and a second attribute that reflects the type of power under which the switch is operating (these attributes are set to the appropriate values in each state's IRM).

Figure 5-4. State transition diagram for telephone switch building.

The State:(Available with Normal Power) has no DRM and only the IRM described above. The Event:*Power fails* causes the transition from the State:(Available With Normal Power) to the State:(Available with Generator Power). For the State:(Available With Generator Power), the IRM makes a probabilistic branch to indicate whether the generator starts (97% likely) or fails to start (3% likely). If the generator starts, the IRM updates the condition of its power supply attribute (to indicate that the “Does the generator start?” question is not to be asked again); otherwise, the Event:*Generator fails* is generated and causes the transition to the State:(Available With Battery Power). The DRM for the State:(Available With Generator Power) uses a delay of 48 hours to represent the fuel tank capacity of the central office’s generator. Then it too generates the Event:*Generator fails*, although this delay time would place us beyond the timeframe assumed for this problem.

The IRM State:(Available With Battery Power) simply sets the object’s attributes to their appropriate values. The DRM executes an 8 hour delay (to represent the reserve capacity of the office’s battery power supply) before generating the Event:*Batteries fail*, which send the object into the State:(Unavailable). The State:(Unavailable) has no DRM (since we are not modeling recovery of telecommunications capabilities), and its IRM simply sets the object’s attributes to their appropriate values. The Event:*Power restored* returns this object to its original State:(Available) from any of the three failed power states.

The state transition diagram for the police station is shown in Figure 5-5. The entire police station model is very similar to the telephone central office model except that the police station does not have a backup battery power supply. This object includes three states, (Available With Normal Power), (Available With Generator Power), and (Unavailable). The police station object has one attribute that reflects the type of power under which the station is operating, and a second attribute to indicate the degree to which it is able to fulfill its assigned duties (based on

the presence or lack of telecommunications and electric power. These attributes are set to the appropriate values in each state's IRM.

Figure 5-5. State transition diagram for police station.

The State:(Available with Normal Power) has no DRM and only the IRM described above. The Event:*Power fails* causes the transition from the State:(Available With Normal Power) to the State:(Available with Generator Power). For the State:(Available With Generator Power), the IRM makes a probabilistic branch to indicate whether the generator starts (95% likely) or fails to start (5% likely). If the generator starts, the IRM updates the condition of its power supply attribute (to indicate that the “Does the generator start?” question is not to be asked again); otherwise, the Event:*Police generator fails* is generated and causes the transition to the State:(Unavailable). The DRM for the State:(Available With Generator Power) uses a delay of 24 hours to represent the fuel tank capacity of the police station’s generator. Then it too generates the Event:*Police generator fails*. The State:(Unavailable) has no DRM (since we are not modeling recovery of the police station), and its IRM simply sets the object’s attributes to their appropriate values. The Event:*Power restored* returns this object to its original State:(Available) from either of the two failed power states.

5.1.3. Graphical Model Solution

One of the main reasons that this problem developed in the way that it did is that, as the first example problem developed, there was no software available to solve the model. For this reason, the model was kept simple enough to solve graphically, and that graphical solution is shown in Figure 5-6. Each path in that figure shows a textual description of each of the various major events that occur on that path. The interactions between the DRMs can be seen through the times at which the different events occur. One can also see the probabilistic branching behavior of both the DRM (whether power is restored after two hours) and the two power consuming

objects' IRMs (generators fail to start). This behavior demonstrates, in a very simple way, the basic concepts involved in the OBEST modeling methodology.

Figure 5-6. Graphical Solution of the police station – electric power problem.

An example problem was constructed to explore the use of DRMs in the OBEST methodology considered the fuel tank supply for a gas burner. In this problem, the flame for a gas burner is controlled by a valve that is opened according to a probabilistic schedule. The valve draws from a fuel system consisting of five tanks (a tank farm), including one 10-gallon tank, one 15-gallon tank, and three 25-gallon tanks. All tanks start out full. Changing from tank to tank occurs without failure, and when all tanks are empty, the problem is over. An illustration of the example problem is shown in Figure 5-7. The objective of the problem is to determine a distribution for how long the fuel supply will last.

The system structure diagram (Figure 5-8) includes two functional blocks, the valve and the fuel tank system. The interaction diagram is shown in Figure 5-9 and the functional interactions for the model are specified in Table 5-2. Interactions occur between the two functional blocks in both directions, as indicated in Figure 5-9. The valve object requests a specified fuel flow from

the fuel system, and the fuel system object provides the fuel supply in response to the valve objects requests. The state transition diagrams for these objects are discussed in the following paragraphs. The discussion includes a description of the object, events, and event transitions, and a narrative of the “if – then – else” scripts and logical conditions for each state’s IRM and DRM. The detailed “if – then – else” scripts and logical conditions are provided in Appendix B.

Figure 5-7. Fuel tank supply for a gas burner.

Figure 5-8. System structure diagram for gas burner fuel supply model.

Figure 5-9. Interaction diagram for gas burner fuel supply model.

Table 5-2. Functional interactions for gas burner fuel supply model.

Valve - Fuel System: The valve requests a specified fuel flow from the fuel system.
Fuel System - Valve: The fuel system provides fuel supply in response to the valve's requests.

5.2.2. The Valve Object

The conditions for requesting a specified fuel flow from the fuel system and turning the valve on and off are embedded within the valve object. The valve object has two states (On and Off) and two events that cause transitions between these states (*Turn On* and *Turn Off*). The state transition diagram for the valve object is shown in Figure 5-10. One attribute is defined for this object, FlowRequest (FR), with two possible values, 0 gallons per minute (gpm) and 2 gpm. The initial state for the valve object is State:(Off) and the initial value for FR is 0 gpm. The analysis is started by generating the Event:*Turn On*.

Figure 5-10. State transition diagram for the valve.

The State:(Off) describes the situation where the valve is closed, no fuel from the tank farm is flowing, that is the FR is 0 gpm, and thus, the gas burner is off. After several possible time periods, the fuel level in the fuel system is evaluated, and if fuel is available, the valve will be turned on. The possible time periods define the schedule for turning on the valve. These time periods are defined as delay times and are probabilistically weighted. The various delay times may be viewed as approximating the demands that have been placed on the system by an outside source. For this example, the probabilistic schedule for turning on the valve is specified in Table 5-3. There is a 0.30 probability of a 10-minute or 30-minute delay time and a 0.40 probability of a 20-minute delay time before the valve is turned on. These conditions are embedded in the IRM

and the DRM for this of valve object. When the valve object is in the State:(Off), the IRM sets the FR to 0 gpm. Additionally, the IRM includes a clock that tracks the time the valve object is in this state. The schedule for turning on the valve is embedded in the DRM for the State:(Off). First the condition of the fuel level in the fuel system is considered. If the fuel level is 0, that is empty, then the valve object does nothing and it remains in the State:(Off). If the fuel level is not 0, then one of the three probabilistically weighted delay times is selected before the valve is turned on and the gas burner is lit. After the selected delay time, the clock is reset, the Event:*Turn On* is generated, and the object transitions from State:(Off) to State:(On).

Table 5-3. Probabilistic schedule for turning on the valve.

Probability	Delay Time
0.30	10
0.40	20
0.30	30

The State:(On) describes the situation where the valve is open, fuel is flowing from the tank farm, that is the FR is 2 gpm, and the gas burner is on. After running the gas burner for several possible time periods, the valve will be turned off again. The probabilistic schedule for turning off the valve is specified in Table 5-4. In this example, there is a 0.20 probability of 10-minute delay, a 0.30 probability of a 20-minute delay, a 0.40 probability of a 30-minute delay, and a 0.10 probability of a 40-minute delay before the valve is turned off.

Table 5-4. Probabilistic schedule for turning off the valve.

Probability	Delay Time
0.20	10
0.30	20
0.40	30
0.10	40

These conditions are embedded in the IRM and DRM for this state of the valve object. When the valve object is in the State:(On), the IRM sets FR to 2 gpm and starts the clock to track the time the valve object is in this state. The schedule for turning off the valve is embedded in the DRM for State:(On). Within the DRM, one of the four delay times is selected before the valve is turned off. After the selected delay time, the clock is reset, the Event:*Turn Off* is generated, and

the object transitions from State:(On) to State:(Off). In this manner, the valve object transitions between State:(Off) and State:(On) until the fuel level in the fuel system is empty.

5.2.3. The Fuel System Object

The conditions for providing the fuel supply in response to requests from the valve object are embedded within the fuel system object. These conditions include switching from one tank to a new tank as the previous one is drained, as well as the possibility of refilling the tanks after they have been drained. The fuel system has four states (Gas, Tank Change, Empty, and Refilling) and five events (*Change*, *Drained*, *Refill*, *Full*, and *New Tank*) that cause transitions between the four states. The state transition diagram for the fuel system object is shown in Figure 5-11. One attribute is defined for this object, FuelLevel (FL), with six possible values (100 gallons (gal), 75 gal, 50 gal, 25 gal, 10 gal, or 0 gal) that correspond to 5, 4, 3, 2, 1, or 0 tanks that contain fuel. The initial state for the fuel system object is State:(Gas), and the initial value of FL is 100. The object includes a clock to track the time between changing tanks.

Figure 5-11. State transition diagram for fuel system.

The State:(Gas) describes the situation where the fuel system is providing fuel in response to the request from the valve object. The response of this object is conditional on the fuel request from the valve object (Valve:FR). The fuel system draws on the three 25-gallon tanks first, then the 15-gallon tank, and finally, the ten-gallon tank. Each tank provides fuel for a specified time period before it is drained and the fuel system has to change to the next tank. These time periods are delay times for the State:(Gas) before the Event:*Change* is generated to cause a transition to the State:(Tank Change). The delay times for changing tanks are specified in Table 5-5. For the 25-gallon tanks, the delay time is 12.5 minutes; for the 15-gallon tank, 7.5 minutes; and for the 10-gallon tank, 5.0 minutes. These conditions are embedded in the IRM and DRM for the State:(Gas). In the IRM, if Valve:FR=0 gpm, then the clock for the fuel system object is

stopped, and no activity occurs. Otherwise, when Valve:FR=2 gpm, the clock for the fuel system is turned on and the object will proceed to the DRM. For each fuel level, the DRM will delay for the corresponding time, then generate the Event:*Change* to cause a transition from this state to the State:(Tank Change). Thus, time is a surrogate for the actual level of depletion within each tank.

Table 5-5. Delay times for changing tanks.

Fuel Level (FL)	Delay Time
100	12.5
75	12.5
50	12.5
25	7.5
10	5.0

The purpose of the State:(Tank Change) is to set the value of FL as the fuel system changes from tank to tank. If the FL=100, the delay time of 12.5 has elapsed, and the Event:*Change* has been generated, then the IRM for the State:(Tank Change) will set the FL=75, reset the clock for the fuel system object, and generate the Event:*New Tank*. The fuel system object will continue to transition between the State:(Gas) and the State:(Tank Change) until the fuel system reaches FL=0, then the Event:*Drained* will be generated to cause the transition from State:(Tank Change) to State:(Empty). There is no DRM for the State:(Tank Change).

As the basic problem for the gas burner fuel supply is defined, when the fuel system reaches the State:(Empty), that is the end of the problem. With the probabilistic schedules defined in the valve object for turning the valve on and off, a set of scenarios can be generated to develop a probability distribution for how long the fuel supply will last. The State:(Refilling) and the events *Refill* and *Full* are included in the state transition diagram for completeness. Currently, the IRM for the State:(Empty) is used to set the FL=0 and to reset the clock. No DRM or conditions for generating the Event:*Refill* are specified. Also, the IRM for the State:(Refilling) is defined to set the FL=100, to reset the clock, and to generate the Event:*Full*. No DRM is defined.

In terms of demonstrating the delay response capabilities of the OBEST method, from this example, it is evident that time can be used a surrogate for the depletion of physical quantities and/or resources, as long as this depletion happens at a constant rate. An alternative, and perhaps better approach would be to develop an explicit depletion model that reflects more accurately the depletion of a specific physical quantity of interest. This type of model could also be incorporated in the OBEST model of the system.

5.3. A Singles Racquetball Game

As work on the project was initiated, the project staff decided that a model of a sports game characterized by a number of fairly straightforward actions and responses between two competitors would provide a good mechanism to investigate the capabilities of the OBEST methodology, as well as to communicate generally the capabilities of the method. A singles racquetball game was selected because of its simplicity and because it provided a mechanism to demonstrate many of the capabilities of the OBEST method.

In a singles racquetball game, two players alternate serving and receiving serve, and making and returning shots. For each point, the server has two opportunities to put the ball into play. All of the play that occurs after the receiver executes a successful return of serve is called a rally. If the server wins the rally (that is the receiver fails to return a server's shot during a rally), one point is scored. A server is entitled to continue serving until an out occurs (that is the server makes two fault serves or fails to return the receiver's shot during the rally). When an out occurs, there is an exchange of serve, and the server becomes the receiver and the receiver becomes the server. A standard game is played to 15 points. In a tournament situation, a referee would monitor the serve exchanges and track the score for the game. In a recreational situation, the players themselves track the status of the game.

A player in a racquetball game has a variety of serve and shot return options. Depending on a player's skill level, a player's choice of serve or shot return is often conditional on a number of factors including an assessment of an opponent's skill level, the immediately previous shot, an opponent's position on the court, an expected return to a given shot or serve, and overall strategy for winning the rally. Generally, serve options are of three basic types: a drive serve, which is a hard, fast, straight serve to the far corner of the court; a lob serve, which has a high arc; and a Z serve, which strikes the front wall, then the side wall at a tight angle, and then angles back to the opposite corner of court. These basic serves, of course, have many variations depending on speed and placement. Basic shots, including serve returns, include four types: a down the line shot, which is a hard, fast, straight shot down either side of the court; a ceiling shot, which generally strikes the ceiling before it hits the front wall; a crosscourt shot, which angles to the opposite side of the court after striking the front wall; a pinch shot, which strikes the front wall near the corner then strikes the side wall at a tight angle. Again, these basic shots have many variations.

The OBEST model of a singles racquetball game was constructed based on the general description in the previous two paragraphs. As this model was constructed, it evolved from a relatively simple model of a rally between two players (with individual shot return percentages) to a more complex model that considered player profiles defined by overall shot return percentages, as well as specific serve returns, with success probabilities that are conditional on the different serve types received from their opponent. Initially, each player had only one serve, and subsequently, a second serve was incorporated into the model. Additionally, the racquetball game itself was modeled as a separate functional block that tracked the status of overall system attributes, namely the score in the game and the serve assignment. Again, this model demonstrates several of the key ideas and capabilities of the OBEST methodology including variable ordering of a set of repetitive events, how objects are related by similar events, and the incorporation of conditional probabilistic information.

5.3.1. System Structure and Interactions

The system structure diagram (Figure 5-12) for the model was defined by three functional blocks, two single player objects, Player A and Player B, and the game object. The interaction diagram is shown in Figure 5-13, and the functional interactions for the model are specified in Table 5-6. Interactions occur among the functional blocks, as indicated in Figure 5-13. The players hit shots and return serves to each other and provide their status to the game. The game tracks serving and scoring for both players. The state transition diagrams for the three functional blocks are discussed in the following paragraphs. The discussion includes a description of the object, events, and event transitions, and a narrative of the “if – then – else” scripts and logical conditions for the IRM and DRM for each state. The detailed “if – then – else” scripts and logical conditions are provided in Appendix C.

Figure 5-12. System structure diagram for model of a singles racquetball game.

Figure 5-13. Interaction diagram for model of singles racquetball game.

Table 5-6. Functional interactions for model of singles racquetball game.

A-B:	Player A hits shot returns and serves to Player B.
A-G:	Player A provides its status to the Game.
B-A:	Player B hits shot returns and serves to Player A.
B-G:	Player B provides its status to the Game.
G-A:	The Game tracks serving and scoring for Player A.
G-B:	The Game tracks serving and scoring for Player B.

5.3.2. Player A Object

The Player A object represents one player in a singles racquetball game. Player A hits shot returns and serves to Player B. Additionally, Player A provides its status to the Game. To model Player A's actions and responses during a racquetball game, five states are defined, as illustrated in the state transition diagram for Player A in Figure 5-14. These states include Stop, Waiting to/for Serve, Waiting to Serve2, Waiting to Return, and Making a Shot. These states are related by a set of events that represents Player A's actions and responses during a racquetball game. These events cause transitions among the five states of the Player A object, as indicated by the directed arrows in Figure 5-14. The set of events defined for the model include: *Start new rally*, *Player A faults 1st serve*, *Player A faults 2nd serve*, *Player A makes shot*, *Player A misses shot*, *Player B makes shot*, *Player B misses shot*, and *Player B faults 2nd serve*.*

Two attributes are defined for this object to provide information about Player A's serve, serve return, and shot return profiles. The first attribute, SHOT_TYPE, defines the serve and return options Player A uses during a game. For this model, eight possible values are defined for the attribute SHOT_TYPE. Four of these values, (Drive, Lob, ZDrive, ZLob), define successful serve options; the value (Fault) is used to define a first fault serve. The value (Out) is used to define either a second fault serve or a failed return; the value (Return) is defined for successful return shots during a rally; and the value (Unset) is defined to set initial conditions. The quantitative values for Player A's SHOT_TYPE are defined by the player's serve and shot return profiles. The serve profile is defined by a set of probability values for first and second serve options. The shot return profile is defined by the player's overall shot return percentage. When Player A is serving, the five applicable values of SHOT_TYPE are (Drive, Lob, ZDrive, ZLob, Fault, Out). The probability values for Player A's serve options are presented in Table 5-7. When Player A is making a shot during a rally, the applicable values are (Return, Out). Player

* The event *Player B faults 1st serve* exists, but is used within the Player B object as it demands no response from Player A.

A's overall shot return profile is 0.80 successful returns and 0.20 failed returns or outs. Note that all of the individual probability values used in this example are for illustration purposes only and are not based on actual "measurements" of real game conditions.

Figure 5-14. State transition diagram for Player A object.

The second attribute for this object, `SERVE_RETURN`, defines the serve return options Player A uses during a game. For this model, five possible values are defined for the attribute `SERVE_RETURN`. Three of these values (Down the line, Ceiling, Crosscourt/Pinch) define successful serve return options. The value (Skip) is used to define a failed serve return, and the value (Unset) is used to set initial conditions. The quantitative values for Player A's serve return profiles are defined by a set of probability values that are conditional on Player B's serve. The values for Player A's serve return are presented in Table 5-8.

The State:(Stop) is defined to reflect the situation when play is stopped before each serve and after each rally during a game. The Event:*Start new rally* causes the transition from the State:(Stop) to the State:(Waiting to/for Serve). No IRM or DRM is defined for the State:(Stop). The Event:*Start new rally* is generated external to the Player A object by the Game object.

Table 5-7. Player A's serve profile.

Serve Option	1 st Serve	2 nd Serve
Drive serve	0.72	0.05
Lob serve	0.03	0.84
ZDrive serve	0.03	0.00
ZLob serve	0.01	0.03
Fault serve	0.21	0.00
Out	0.00	0.08

Table 5-8. Player A's serve return profile conditional on Player B's serve.

Player A's Serve Return Options	Player B's Serve			
	Drive Serve	Lob Serve	ZDrive Serve	ZLob Serve
Down the line	0.45	0.25	0.40	0.20
Ceiling	0.35	0.60	0.30	0.50
Crosscourt/Pinch	0.05	0.15	0.05	0.25
Skip	0.15	0.05	0.25	0.05

The State:(Waiting to/for Serve) is used to describe the situation where Player A is waiting to take the first serve of a point or waiting to receive Player B's first and second serves. If Player A is serving, the serve options will reflect the profile indicated in the column "1st Serve" in Table 5-7. This information is used to set the values (Drive, Lob, ZDrive, ZLob, Fault, Out) used for the SHOT_TYPE attribute for this state. That is, 72% of the time, Player A will execute a drive serve; 3% of the time, a lob serve; 3% of the time, a ZDrive serve; and 1% of the time, a ZLob serve. In the situation where Player A executes a successful serve, the Event:*Player A makes shot*, will be generated by the object. This will cause a transition for Player A from the State:(Waiting to/for Serve) to the State:(Waiting to Return). According to Player A's serve profile in Table 5-7, 21% of the time, Player A will fail to execute a successful first serve. In this case, the Event:*Player A faults 1st serve* will be generated by the object, and Player A will transition from the State:(Waiting to/for Serve) to the State:(Waiting to Serve2). In the situation where Player A is waiting for Player B's serve, two events can occur. If Player B executes a

successful first or second serve, then the Event:*Player B makes shot* is generated, and Player A will transition from the State:(Waiting to/for Serve) to State: (Making a Shot). Otherwise, Player A will stay in the State:(Waiting to/for Serve) until Player B faults a second serve. Then the Event:*Player B faults 2nd serve* is generated, and Player A will transition from the State: (Waiting to/for Serve) to the State:(Stop). The conditions for this state are embedded in the IRM. No delayed responses are defined and therefore no DRM is defined for this state.

The State:(Waiting to Serve2) describes the situation where Player A has faulted a first serve and is waiting to make a second serve. When Player A is serving as second serve, the serve options will reflect the profile indicated in the column “2nd Serve” in Table 5-7. This information is used to set the values used for the SHOT_TYPE attribute for this state. That is, 5% of the time, Player A will execute a drive serve; 84% of the time, a lob serve; 0% of the time, a ZDrive serve; and 3% of the time, a ZLob serve. In the situation where Player A executes a successful second serve, the Event:*Player A makes shot*, will be generated by the object. This will cause a transition for Player A from the State:(Waiting to/for Serve) to the State:(Waiting to Return). According to Player A’s serve profile in Table 5-7, 8% of the time, Player A will fail to execute a successful second serve. In this case, the Event:*Player A faults 2nd serve* will be generated by the object, and Player A will transition from the State:(Waiting to Serve2) to the State:(Stop). The conditions for this state are embedded in the IRM. No delayed responses are defined and therefore no DRM is defined for this state.

The State:(Waiting to Return) is used to describe the situation where Player A is waiting to execute a serve return or a shot return during a rally. This will occur in a game after Player A has executed a successful first or second serve or a successful shot return, that is after the Event:*Player A makes a shot* has been generated. No IRM or DRM is defined for this state. It is a waiting state, in which Player A will remain until some action from Player B occurs. Two possible actions from Player B are considered; either Player B makes a shot and the rally will continue, or Player B misses a shot and Player A wins the rally. When the Player B object generates the Event:*Player B makes shot*, then Player A will transition from the State:(Waiting to Return) to the State:(Making a Shot). When the Player B object generates the Event:*Player B misses shot*, then Player A will transition from the State:(Waiting to Return) to the State:(Stop).

The State:(Making a Shot) describes the conditions for Player A’s serve returns or shot returns. If Player A is serving for the point, then this state is used to define the conditions for Player A’s shot return profile. For this model, a player’s shot return profile is defined by an overall return percentage. No conditional statements are defined for a player’s shot returns, and the overall return percentage is used for all the player’s shot returns, other than serve returns, during a rally. As defined above, Player A’s shot return profile is 80% successful shot returns and 20% unsuccessful shot returns. This information is used to set the values (Return, Out) used for the SHOT_TYPE attribute for this state. If Player B is serving for the point, then this state is used to define Player A’s serve return options, which are conditional on the type of serve Player B executes. As indicated in Table 5-8, Player A has four different serve return profiles dependent on four different types of serves Player B might execute. For example, if Player B executes a drive serve, then 45% of the time, Player A will take a down the line return shot, 35% of the time a ceiling shot, and 5% of the time a crosscourt/pinch shot. According to Table 5-8, Player A will fail to execute, or skip, a return to Player B’s drive serve 15% of the time. This information, along with the other information in Table 5-8 on Player A’s returns to Player B’s different

serves, are used to define the values for the attribute `SERVE_RETURN` that are used for this state. For either a shot return or a serve return, if Player A executes a successful return, then the rally continues; if not, Player B wins the rally. That is, if Player A executes a successful shot or serve return, then the Event:*Player A makes shot* is generated, and Player A transitions from the State:(Making a Shot) to the State:(Waiting to Return). If Player A does not execute a successful shot or serve return, then the Event:*Player A misses shot* is generated, and Player A transitions from the State:(Making a Shot) to the State:(Stop).

5.3.3. Player B Object

The Player B object represents a second player in a singles racquetball game. Player B hits shot returns and serves to Player A. Additionally, Player B provides its status to the Game. To model Player B's actions and responses during a racquetball game, five states are defined, as illustrated in the state transition diagram for Player B in Figure 5-15. The state transition diagram for Player B is very similar to that for Player A. The states and events are basically the same, but the events differ to reflect actions and responses from Player B's point of view. The states include Stop, Waiting to/for Serve, Waiting to Serve2, Waiting to Return, and Making a Shot. These states are related by a set of events that represents Player B's actions and responses during a racquetball game. These events cause transitions among the five states of the Player B object, as indicated by the directed arrows in Figure 5-15. The set of events defined for the model include: *Start new rally, Player B faults 1st serve, Player B faults 2nd serve, Player B makes shot, Player B misses shot, Player A makes shot, Player A misses shot, and Player A faults 2nd serve.*

The two attributes, `SHOT_TYPE` and `SERVE_RETURN`, are also defined for this object to provide information about Player B's serve, serve return, and shot return profiles. The attribute, `SHOT_TYPE`, defines the serve and return options Player B uses during a game. For this model, eight possible values are defined for the attribute `SHOT_TYPE`. Four of these values, (Drive, Lob, ZDrive, ZLob), define successful serve options; the value (Fault) is used to define a first fault serve. The value (Out) is used to define either a second fault serve or a failed return; the value (Return) is defined for successful return shots during a rally; and the value (Unset) is defined to set initial conditions. The quantitative values for Player B's `SHOT_TYPE` are defined by the player's serve and shot return profiles. The serve profile is defined by a set of probability values for first and second serve options. The shot return profile is defined by the player's overall shot return percentage. When Player B is serving, the five applicable values of `SHOT_TYPE` are (Drive, Lob, ZDrive, ZLob, Fault, Out). The probability values for Player B's serve options are presented in Table 5-9. When Player B is making a shot during a rally, the applicable values are (Return, Out). Player B's overall shot return profile is 0.75 successful returns and 0.25 failed returns or outs.

The second attribute for this object, `SERVE_RETURN`, defines the serve return options Player B uses during a game. For this model, five possible values are defined for the attribute `SERVE_RETURN`. Three of these values (Down the line, Ceiling, Crosscourt/Pinch) define successful serve return options. The value (Skip) is used to define a failed serve return, and the value (Unset) is used to set initial conditions. The quantitative values for Player B's serve return profiles are defined by a set of probability values that are conditional on Player A's serve. The values for Player B's serve return are presented in Table 5-10.

Figure 5-15. State transition diagram for Player B object.

Table 5-9. Player B's serve profile.

Serve Option	1 st Serve	2 nd Serve
Drive serve	0.63	0.03
Lob serve	0.05	0.80
ZDrive serve	0.05	0.00
ZLob serve	0.02	0.02
Fault serve	0.25	0.00
Out	0.00	0.05

Table 5-10. Player B's serve return profile conditional on Player A's serve.

Player B's Serve Return Options	Player A's Serve			
	Drive Serve	Lob Serve	ZDrive Serve	ZLob Serve
Down the line	0.48	0.18	0.45	0.30
Ceiling	0.26	0.64	0.15	0.40
Crosscourt/Pinch	0.08	0.16	0.02	0.20
Skip	0.18	0.02	0.33	0.10

The State:(Stop) is defined to reflect the situation when play is stopped before each serve and after each rally during a game. The Event:*Start new rally* causes the transition from the State:(Stop) to the State:(Waiting to/for Serve). No IRM or DRM is defined for the State:(Stop). The Event:*Start new rally* is generated external to the Player B object by the Game object.

The State:(Waiting to/for Serve) is used to describe the situation where Player B is waiting to take the first serve of a point or waiting to receive Player B's first and second serves. If Player B is serving, the serve options will reflect the profile indicated in the column "1st Serve" in Table 5-9. This information is used to set the values (Drive, Lob, ZDrive, ZLob, Fault, Out) used for the SHOT_TYPE attribute for this state. That is, 63% of the time, Player B will execute a drive serve; 5% of the time, a lob serve; 5% of the time, a ZDrive serve; and 2% of the time, a ZLob serve. In the situation where Player B executes a successful serve, the Event:*Player B makes shot*, will be generated by the object. This will cause a transition for Player B from the State:(Waiting to/for Serve) to the State:(Waiting to Return). According to Player B's serve profile in Table 5-9, 25% of the time, Player B will fail to execute a successful first serve. In this case, the Event:*Player B faults 1st serve* will be generated by the object, and Player B will transition from the State:(Waiting to/for Serve) to the State:(Waiting to Serve2). In the situation where Player B is waiting for Player A's serve, two events can occur. If Player A executes a successful first or second serve, then the Event:*Player A makes shot* is generated, and Player B will transition from the State:(Waiting to/for Serve) to State: (Making a Shot). Otherwise, Player B will stay in the State:(Waiting to/for Serve) until Player A faults a second serve. Then the Event:*Player A faults 2nd serve* is generated, and Player B will transition from the State: (Waiting to/for Serve) to the State:(Stop). The conditions for this state are embedded in the IRM. No delayed responses are defined and therefore no DRM is defined for this state.

The State:(Waiting to Serve2) describes the situation where Player B has faulted a first serve and is waiting to make a second serve. When Player B is serving as second serve, the serve options will reflect the profile indicated in the column "2nd Serve" in Table 5-9. This information is used to set the values used for the SHOT_TYPE attribute for this state. That is, 3% of the time, Player A will execute a drive serve; 80% of the time, a lob serve; 0% of the time, a ZDrive serve; and

2% of the time, a ZLob serve. In the situation where Player B executes a successful second serve, the Event:*Player B makes shot*, will be generated by the object. This will cause a transition for Player B from the State:(Waiting to/for Serve) to the State:(Waiting to Return). According to Player B's serve profile in Table 5-9, 5% of the time, Player B will fail to execute a successful second serve. In this case, the Event:*Player B faults 2nd serve* will be generated by the object, and Player B will transition from the State:(Waiting to Serve2) to the State:(Stop). The conditions for this state are embedded in the IRM. No delayed responses are defined and therefore no DRM is defined for this state.

The State:(Waiting to Return) is used to describe the situation where Player B is waiting to execute a serve return or a shot return during a rally. This will occur in a game after Player B has executed a successful first or second serve or a successful shot return, that is after the Event:*Player B makes a shot* has been generated. No IRM or DRM is defined for this state. It is a waiting state, in which Player B will remain until some action from Player A occurs. Two possible actions from Player A are considered; either Player A makes a shot and the rally will continue, or Player A misses a shot and Player B wins the rally. When the Player A object generates the Event:*Player A makes shot*, then Player B will transition from the State:(Waiting to Return) to the State:(Making a Shot). When the Player A object generates the Event:*Player A misses shot*, then Player B will transition from the State:(Waiting to Return) to the State:(Stop).

The State:(Making a Shot) describes the conditions for Player B's serve returns or shot returns. If Player B is serving for the point, then this state is used to define the conditions for Player B's shot return profile. For this model, a player's shot return profile is defined by an overall return percentage. No conditional statements are defined for a player's shot returns, and the overall return percentage is used for all the player's shot returns, other than serve returns, during a rally. As defined above, Player B's shot return profile is 80% successful shot returns and 20% unsuccessful shot returns. This information is used to set the values (Return, Out) used for the SHOT_TYPE attribute for this state. If Player A is serving for the point, then this state is used to define Player B's serve return options, which are conditional on the type of serve Player A executes. As indicated in Table 5-10, Player B has four different serve return profiles dependent on four different types of serves Player A might execute. For example, if Player A executes a drive serve, then 48% of the time, Player B will take a down the line return shot, 26% of the time a ceiling shot, and 8% of the time a crosscourt/pinch shot. According to Table 5-10, Player B will fail to execute, or skip, a return to Player A's drive serve 18% of the time. This information, along with the other information in Table 5-8 on Player B's returns to Player A's different serves, are used to define the values for the attribute SERVE_RETURN that are used for this state. For either a shot return or a serve return, if Player B executes a successful return, then the rally continues; if not, Player A wins the rally. That is, if Player B executes a successful shot or serve return, then the Event:*Player B makes shot* is generated, and Player B transitions from the State:(Making a Shot) to the State:(Waiting to Return). If Player B does not execute a successful shot or serve return, then the Event:*Player B misses shot* is generated, and Player B transitions from the State:(Making a Shot) to the State:(Stop).

5.3.4. The Game Object

For this model, the Game object is defined to track the overall status of the game as it progresses, that is, to track serving and scoring for the Player A and the Player B objects. The states, events,

and attributes for the Game object reflect aspects of a racquetball game occur during a change in serve. These include serve assignments, starting and ending a rally, which player wins a rally, and scoring. To model the progress of a racquetball game, five states are defined, as illustrated in the state transition diagram for the Game in Figure 5-16. The states include Idle, Starting Game, Waiting, Player A Wins Rally, and Player B Wins Rally. These states are related by a set of events that represents the Game's actions and responses, as well as the players' actions associated with a change of serve. These events cause transitions among the five states of the Game object, as indicated by the directed arrows in Figure 5-16. The set of events defined for the model include: *Start game*, *Start new rally*, *Player A misses shot*, *Player A faults 2nd serve*, *Player B misses shot*, *Player B faults 2nd serve*, and *End game*.

Figure 5-16. State transition diagram for Game Object.

Five attributes are defined for this object to provide information about the status of the game. The attribute **SERVE** has two possible values, (A, B) and is used to track which player is serving a point. For the initial conditions, **SERVE**=(A). The attribute **SHOT** has three possible values, (A, B, 0), and is used to track which player wins a shot. The value **SHOT**=(0) is set for the

initial conditions. The attribute SCORE_A has integer values 0 to 15 and is used to track Player A's score during the game. For the initial conditions, SCORE_A=(0). The attribute SCORE_B has integer values 0 to 15 and is used to track Player B's score during the game. For the initial conditions, SCORE_B=(0). The attribute WINNER has three possible values, (A, B, Unknown), and is used to define the game winner. For the initial conditions, WINNER=(Unknown).

The State:(Idle) is used to represent the situation before the game is initiated and after the game is concluded. The Event:*Start game* is generated external to the Game object to initiate a game, and causes the transition from the State:(Idle) to the State:(Starting Game). No IRM or DRM is defined for the State:(Idle).

The State:(Starting Game) is used to set the values of each of the attributes for the initial game conditions. The initial conditions are set by the IRM for the state, the Event:*Start new rally* is generated, and the Game object transitions from the State:(Starting Game) to the State:(Waiting). There are no delayed responses for this state, so no DRM is specified.

The State:(Waiting) is used to describe the situation where Player A and Player B are engaged in a rally and the Game is waiting for the rally to conclude. No IRM or DRM is defined for this state. It is a waiting state, in which the Game object will remain until some specified action from Player A or Player B ends the rally. Two possible actions from Player A will end a rally; either Player A misses a shot or Player A faults a second serve. When the Player A object generates the Event:*Player A misses shot*, or the Event:*Player A faults 2nd serve*, then the Game object will transition from the State:(Waiting) to the State:(Player B Wins Rally). The similar two actions from Player B will end a rally; either Player B misses a shot or Player B faults a second serve. When the Player B object generates the Event:*Player B misses shot* or the Event:*Player B faults 2nd serve*, then the Game object will transition from the State:(Waiting) to the State:(Player A Wins Rally).

The State:(Player A Wins Rally) is used to update the scoring, set the winner of the shot, and set the serve based on the conditions of the immediately preceding point. The related attributes are updated in the IRM. If Player A was serving (SERVE=(A)) and had 14 points (SCORE_A=(14)) during the immediately preceding point, then by winning the rally, Player A would win the game. The Game object will increment Player A's score (SCORE_A=(15)), declare Player A the winner (WINNER=(A)), and generate the Event:*End game*. If Player A was serving and did not have 14 points during the immediately preceding point, then the Game object will increment Player A's score accordingly, declare Player A as the winner of the point (SHOT=(A)), and generate the Event:*Start new rally*. The remaining situation occurs when Player A does not serve the point (SERVE=(B)), and by winning the rally, Player A gets the serve. In this case, the Game object will declare Player A the winner of the point (SHOT=(A)), assign the serve to Player A (SERVE=(A)), and generate the Event:*Start new rally*. No delayed responses, and therefore no DRM are defined for this state.

The State:(Player B Wins Rally) is also used to update the scoring, set the winner of the shot, and set the serve based on the conditions of the immediately preceding point. The related attributes are updated in the IRM. If Player B was serving (SERVE=(B)) and had 14 points (SCORE_B=(14)) during the immediately preceding point, then by winning the rally, Player B would win the game. The Game object will increment Player B's score (SCORE_B=(15)),

declare Player B the winner (WINNER=(B)), and generate the Event:*End game*. If Player B was serving and did not have 14 points during the immediately preceding point, then the Game object will increment Player B's score accordingly, declare Player B as the winner of the point (SHOT=(B)), and generate the Event:*Start new rally*. The remaining situation occurs when Player B does not serve the point (SERVE=(A)), and by winning the rally, Player B gets the serve. In this case, the Game object will declare Player B the winner of the point (SHOT=(B)), assign the serve to Player B (SERVE=(B)), and generate the Event:*Start new rally*. No delayed responses are defined for this state, and therefore no DRM is specified.

5.3.5. Modeling Application and Alternatives

Section 5 describes the model of the racquetball game as it was developed for this project. In an actual racquetball game, a skillful player will execute each shot conditional on a number factors, including an assessment of an opponent's skill level, the immediately previous shot, an opponent's position on the court, an expected return to a given shot or serve, and overall strategy for winning the rally. An opponent's position on the court is an important condition for defining shot placement, for example crosscourt shots. Another important consideration in a racquetball game that is not incorporated in the model is whether a player is left-handed or right-handed, and whether a player places shots to the opponent's backhand or forehand. This model could be applied to generate a number of different scenarios for games between two players with the specified serve, serve return, and shot return profiles. The sequence of events that would occur, the attribute values associated with each event in the sequence, as well as the probabilities for which player would win will differ from game to game.

This model demonstrates how the OBEST method can be applied for systems that are characterized by states, events, and associated attributes that reoccur in variable sequences. Because waiting states were explicitly defined and no delayed responses were defined, this model focused on IRMs rather than DRMs. Additionally, this model demonstrated how one component in a system generates actions and responses based on events generated from another component.

5.4. *Circuit Analysis for Nuclear Power Plant Cable Fires*

As work on the methodology progressed, the project team investigated possible areas of application for ongoing projects in the risk and reliability area. One area of application for which an example problem was developed is circuit analysis for nuclear power plant cable fires. Currently, Sandia is involved in a project for the U.S. Nuclear Regulatory Commission (NRC) to look at the potential effects on systems operations of fires in the cable trays of a nuclear power plant. The cables in these cable trays are bundles of numerous conductors that connect the equipment in a nuclear power plant. As part of this effort for the NRC, extensive circuit analysis is being done manually to map the cables, conductors, and equipment at nuclear power plants for normal operation and then for possible conditions in the event of a cable fire. It was proposed that the OBEST methodology be applied to demonstrate how this process of circuit analysis could be automated.

To develop an OBEST system model for this application, the following approach was taken. First, a specific circuit of interest is defined as a set of partial circuits that reflect the connections of plant equipment through conductors in the cables running through a cable tray. Then, the possible conditions of connectivity between pairs of partial circuits are established to reflect possible cable connections in the event of a fire. In the event of a fire in a cable tray, the possible conditions of connectivity for a path between two partial circuits include the following:

- closed, the condition of normal operation in which a conductor in a cable has not been affected by the fire;
- open, the condition in which a conductor in a cable has been severed as a result of the fire;
- short to positive (+), the condition in which a conductor in a cable has been shorted to the positive power supply;
- short to negative (-), the condition in which a conductor in a cable has been shorted to ground; and
- short to each other, the condition in which a conductor in a cable has been shorted to another conductor within the same cable bundle.

After the possible conditions on connectivity between the partial circuits have been established, then, for each of the possible connections, the conditions of current flow in a circuit are inferred. The possible conditions of current flow include normal, none, short circuit, and unknown. Additionally, for short circuits, the maximum current potential is evaluated to determine possible effects of short circuits on system operations.*

For this approach, a partial circuit is defined as a series connection of one or more circuit elements and their possible connection to a power supply. The elements of a partial circuit include loads, conductors, switches, and power supplies. The load element represents loads in a circuit, such as solenoid coils, indicators, or motors, that can draw electric current and can be connected directly between the + and - voltage power supply terminals without causing a short circuit. The conductor element represents conductors that connect other elements of the circuit with a zero voltage drop in normal operation; if a conductor is connected directly between + and - voltage sources, a short circuit results. The switch element in a partial circuit can act either as an open circuit or as a conductor in normal operation. The power supply element of a partial circuit represents the + or - design voltage for a partial circuit. For this analysis, the power supply is assumed to provide enough current to power the loads in the circuit (in the absence of short circuits).

* The OBEST cable fire model could have been implemented equally well using the DYLAM dynamic event tree method. If this were done, the object model from this section would be replaced with a conventional circuit analysis package driven through a DYLAM front end. The DYLAM analysis would produce results that embody continuous parameter values (voltages, currents, etc.) as opposed to the discrete values required by the OBEST methodology.

The specific circuit of interest used for this example is the solenoid-operated valve control circuit illustrated in Figure 5-17. For this model, this circuit is defined as a set of six partial circuits. The first partial circuit (C_1) includes the switch element 1-HS-3612 and its connection to the + power supply. The second partial circuit (C_2) includes a load element, the solenoid valve 1SV612, and its connection to the - supply. The third partial circuit (C_3) includes a load element, the G, or green indicator, and its connection to the + power supply. The fourth partial circuit (C_4) includes switch element, 1ZS612, indicated closed, and its connection to the - power supply. The fifth partial circuit (C_5) includes a load element, the R, or red indicator, and its connection to the + power supply. The sixth partial circuit (C_6) includes switch element, 1ZS612, indicated open, and its connection to the - power supply. Of interest are the conditions of connectivity and current in the conductors between partial circuits 1 and 2 (path 1-2), partial circuits 3 and 4 (path 3-4), and partial circuits 5 and 6 (path 5-6).

Figure 5-17. Example solenoid-operated valve control circuit.

In this model, a circuit completer is defined to establish the possible conditions of connectivity between partial circuits. The number of partial circuits and paths between them are defined when the model of the circuit of interest is developed. For this example, the circuit of interest is initially comprised of six partial circuits with three paths (1-2, 3-4, and 5-6) between each of two partial circuits. Again, the possible conditions of connectivity include closed, open, short to +, short to -, and short to each other.

In this model, a circuit analyzer is defined to determine the conditions of current flow in a circuit. The conditions of current flow are determined for each path between partial circuits in the circuit of interest and are based on the conditions in the partial circuit and the corresponding circuit completer. For this example, the circuit of interest is comprised of six partial circuits with three paths (1-2, 3-4, and 5-6) between each of two partial circuits. Again, the possible conditions of current flow include normal, none, short circuit, and unknown. For short circuits, the maximum current potential is also evaluated.

5.4.1. System Structure and Interactions

The system structure diagram (Figure 5-18) for the circuit analysis model is defined by seven functional blocks. The load (L), conductor (W), switch (S), and power supply (P_i) objects are elements of a partial circuit (C_n). The other two objects of the model include the circuit analyzer (CA) and the circuit completer (CC). The Circuit Analyzer is not a “physical” object. Rather, it arbitrates among the various physical components to determine the collective behavior of the overall circuit.

Figure 5-18. System structure diagram for circuit analysis.

The interaction diagram is shown in Figure 5-19, and the functional interactions for the model are specified in Table 5-11. Interactions occur among the functional blocks, as indicated in Figure 5-19. The circuit analyzer provides the circuit completer with information on current flow through the conductors between partial circuits. The circuit completer provides the circuit analyzer with information on physical connections and electric potential for the conductors between partial circuits. The circuit analyzer provides to the partial circuit information on current flow through partial circuit n . Partial circuit n provides information about current flow to its power supply, load, conductor, and switch. The load, conductor, switch and power supply provide to partial circuit n information on its physical connections and electric potential drop.

The state transition diagrams for the seven functional blocks are discussed in the following paragraphs. The discussion includes a description of the object, events, and event transitions,

and a narrative of the “if – then – else” scripts and logical conditions for the IRM and DRM for each state. The detailed “if – then – else” scripts and logical conditions are provided in Appendix D.

Table 5-11. Functional interactions for circuit analysis model.

L-C _n :	A load provides to its partial circuit information on its physical connections and electric potential drop.
W-C _n :	A conductor provides to its partial circuit information on its physical connections and electric potential drop.
S-C _n :	A switch provides to its partial circuit information on its physical connections and electric potential drop.
P _n -C _n :	Power supply n provides to its partial circuit information about physical connections and electric potential.
C _n -L:	The partial circuit provides to its loads information about current flow.
C _n -W:	The partial circuit provides to its conductors information about current flow.
C _n -S:	The partial circuit provides to its switches information about current flow.
C _n -P _n :	Partial circuit n provides to its power supply information about current flow.
C _n -CA:	Partial circuit n provides to the circuit analyzer information on its physical connections and electric potential.
CA-C _n :	The circuit analyzer provides to the partial circuit information on current flow through the partial circuit.
CA-CC:	The circuit analyzer provides to the circuit completer information on current flow through the conductors between partial circuits.
CC-CA:	The circuit completer provides to the circuit analyzer information on physical connections for the conductors between partial circuits.

Figure 5-19. Interaction diagram for circuit analysis model.

5.4.2. Load Object

The Load Object (L) represents an element in a partial circuit that draws electric current, for example solenoid coils, indicators, or motors, and can be connected directly between + and - voltage without causing a short circuit. A load provides to its partial circuit information on how it is connected in the circuit, that is, its physical connections and electric potential drop. In turn, a load will receive information from its partial circuit about current flow.

To model the Load Object, one state, Load, is defined as indicated in the state transition diagram in Figure 5-20. No state transitions are defined for this object; events external to this object will cause changes in object attributes. Two attributes are defined for this object to provide to its partial circuit information on its physical connections and electric potential drop within the circuit. The first attribute, CONNECTIVITY, defines the physical connection of a load within its partial circuit. Two values (Conducting, Not Conducting) are defined for this attribute. The second attribute, V-DROP, defines the electric potential drop of the load within the circuit. Three values (0, Operating, Open) are defined for this attribute.

For a circuit of interest, the analyst must define loads as elements of the partial circuits that comprise it. Initial conditions are assumed to be those for normal operation (CONNECTIVITY=Conducting, V-DROP=Operating) for each load defined by the analyst. Three of the six partial circuits defined for the solenoid-operative valve control circuit include a load element: the G and R indicators in C₁ and C₃, respectively; and the solenoid valve in C₂.

Figure 5-20. State transition diagram for the Load Object.

5.4.3. Conductor Object

The Conductor Object (W)^{*} is an element in a partial circuit that represents conductors that connect other elements of the circuit with a zero voltage drop in normal operation; if a conductor is connected directly between + and - voltage sources, a short circuit results. A conductor provides to its partial circuit information on how it is connected in the circuit, that is, its physical connections and electric potential drop. In turn, a conductor will receive information from its partial circuit about current flow.

To model the Conductor Object, one state, Conduct, is defined as indicated in the state transition diagram in Figure 5-21. No state transitions are defined for this object; events external to this object will cause changes in object attributes. Two attributes are defined for this object to provide to its partial circuit information on its physical connections and electric potential drop within the circuit. The first attribute, CONNECTIVITY, defines the physical connection of a conductor within its partial circuit. Two values (Conducting, Not Conducting) are defined for this attribute. The second attribute, V-DROP, defines the electric potential drop of the conductor within the circuit. Two values (0, Open) are defined for this attribute.

Figure 5-21. State transition diagram for the Conductor Object.

For a circuit of interest, the analyst must define conductors as elements of the partial circuits that comprise it. Initial conditions are assumed to be those for normal operation

^{*} The symbol W was selected to represent a conductor because the most common type of conductor is a wire, and because the symbol C is being used to represent a partial circuit.

(CONNECTIVITY=Conducting, V-DROP=0) for each conductor defined by the analyst. Each of the six partial circuits defined for the solenoid-operative valve control circuit could be, but is not necessarily defined to include a conductor between the power supply and its respective load or switch. It would be necessary to include a conductor as an element of a partial circuit that had a power supply as its only other element.

5.4.4. Switch Object

The Switch Object (S) represents an element in a partial circuit that can act as an open circuit or a conductor in normal operation. A switch provides to its partial circuit information on how it is connected in the circuit, that is, its physical connections and electric potential drop. In turn, a switch will receive information from its partial circuit about current flow.

To model the operation of the Switch Object, two states are defined as illustrated in the state transition diagram in Figure 5-22. These states include Open and Close and are related by a set of events that represents the switch's operation. These events cause transitions among the two states of the switch object, as indicated by the directed arrows in Figure 5-22. The set of events defined for the object include *Switch closes*, and *Switch opens*.

Figure 5-22. State transition diagram for the Switch Object.

Two attributes are defined for this object to provide to its partial circuit information on its physical connections and electric potential drop within the circuit. The first attribute, CONNECTIVITY, defines the physical connection of a switch within its partial circuit. Two values (Conducting, Not Conducting) are defined for this attribute. The second attribute, V-DROP, defines the electric potential drop of the switch within the circuit. Two values (0, Open) are defined for this attribute.

For a circuit of interest, the analyst must define switches as elements of the partial circuits that comprise it. Initial conditions are assumed to be those for normal operation, either open or closed, for each switch defined by the analyst. Three of the six partial circuits defined for the solenoid-operative valve control circuit include a switch element: the switch element,

1-HS-3612, indicated open, in C_1 ; the switch element, 1ZS612, indicated closed, in C_4 ; and the switch element, 1ZS612, indicated open, in C_6 .

The State:(Open) is used to describe a switch that is open during normal operation, such as the switch elements in C_1 and C_6 . The initial conditions for an open switch are defined as S:CONNECTIVITY=(Not Conducting) and S:V-DROP=Open. These conditions are embedded in the IRM for this state. The Event:*Switch closes* is generated from outside the circuit system, for example, by a human operator, and causes a transition from the State:(Open) to the State:(Closed).

The State:(Closed) is used to describe a switch that is closed during normal operation, such as the switch element in C_4 . The initial conditions for a closed switch are defined as S:CONNECTIVITY=(Conducting) and S:V-DROP=0. These conditions are embedded in the IRM for this state. The Event:*Switch opens* is generated from outside the circuit system, for example, by a human operator, and causes a transition from the State:(Closed) to the State:(Open).

5.4.5. Power Supply Object

The Power Supply Object (P_i) is an element in a partial circuit that represents the + or – design voltage for C_n . The power supply provides to its partial circuit information about physical connections and polarity. In turn, a power supply will receive information from its partial circuit about current flow.

To model the operation of the Power Supply Object, two states are defined as illustrated in the state transition diagram in Figure 5-23. These states include Provides Power and Does Not Provide Power, and are related an event, *Fuse blows*, that represents the power supply's operation. This event causes a transition between the two states of the power supply object, as indicated by the directed arrow in Figure 5-23. This model does not consider the replacement of a blown fuse. If it did, it would be represented by a separate event, state transition, and arrow.

Figure 5-23. State transition diagram for the Power Supply Object.

Two attributes are defined for this object to provide to its partial circuit information on its physical connections and polarity within the circuit. The first attribute, CONNECTIVITY,

defines the physical connection of a power supply within its partial circuit. Two values (Conducting, Not Conducting) are defined for this attribute. The second attribute, POLARITY, defines the polarity of the power supply. Two values (+, -) are defined for this attribute.

For a circuit of interest, the analyst must define power supplies as elements of the partial circuits that comprise it. A circuit model may include one or more power supplies that provide power for one or more partial circuits. Initial conditions are assumed to be normal operation for design voltage for each power supply defined by the analyst. Two power supplies are defined for the solenoid-operative valve control circuit. P_1 is the positive power supply and an element of C_1 , C_3 , and C_5 . P_2 is the negative power supply and an element of C_2 , C_4 , and C_6 .

The State:(Provides Power) is used to describe a power supply during normal operation. The initial conditions for the two power supplies in the solenoid-operative valve control circuit are defined as $P_1:CONNECTIVITY = (\text{Conducting})$, $P_1:POLARITY = (+)$, and $P_2:CONNECTIVITY = (\text{Conducting})$, $P_2:POLARITY = (-)$, respectively. These conditions are embedded in the IRM for this state. The Event:*Fuse blows* will be generated from the partial circuit object and causes a transition from the State:(Provides Power) to the State:(Does Not Provide Power).

The State:(Does Not Provide Power) is used to describe a power supply that has a blown fuse and is not providing current to power the loads in its partial circuit. In this state, the connectivity is set to $P_1:CONNECTIVITY = (\text{Not Conducting})$. These conditions are embedded in the IRM for this state.

5.4.6. Partial Circuit Object

The Partial Circuit Object (C_n) represents a series connection of one or more circuit elements and their possible connection to a power supply. The partial circuit provides information about current flow to its power supply, loads, conductors, and switches. The circuit analyzer provides to the partial circuit information on current flow through the partial circuit.

To model the Partial Circuit Object, one state, Determine C_n Elements & Attributes, is defined as indicated in the state transition diagram in Figure 5-24. No state transitions are defined for this object; events external to this object will cause changes in object attributes. Two attributes, POLARITY and CONDUCTION, are defined for this object to provide information on the polarity and conduction of the partial circuit. The values of these attributes are determined by input from the associated circuit elements.

**Determine C_n
Elements &
Attributes**

Figure 5-24. State transition diagram for the Partial Circuit Object.

The first attribute, POLARITY, defines the polarity (+ or -) of the partial circuit based on the polarity of the corresponding power supply. Three values (+, -, Not Determined) are defined for this attribute. The second attribute, CONDUCTION, defines the conduction in the partial circuit. Four values (Open, Direct, Through Load, Unknown) are defined for this attribute. The initial conditions for a partial circuit are set to await information from circuit elements (C_n :POLARITY=(Not Determined) and C_n :CONDUCTION=(Unknown)).

Conditions to set the attribute values for a partial circuit are embedded in the IRM for the State:(Determine C_n Elements & Attributes). If the circuit analyzer generates a short circuit current for the partial circuit, and the connectivity of the power supply for this partial circuit is conducting (P_i :CONNECTIVITY=Conducting), then the Event:*Fuse blows* will be generated. Next, if the polarity of the partial circuit is not determined (C_n :POLARITY=(Not Determined)), then it will be set to the same polarity of the power supply for the partial circuit. If the connectivity of any of the circuit elements is (Not Conducting), then the conduction for the partial circuit will be set to open (C_n :CONDUCTION=(Open)). If the electric potential of the loads in the partial circuit is set for normal operating conditions, then the conduction for the partial circuit is through the loads (C_n :CONDUCTION=(Through Load)). Finally, if none of the previous conditions apply, then the conduction is defined as direct through the partial circuit (C_n :CONDUCTION:(Direct)).

5.4.7. Circuit Completer Object

The purpose of the Circuit Completer Object (CC) is to establish all possible conditions of connectivity in a set of partial circuits. The circuit completer provides the circuit analyzer with information on physical connections and electric potential for the conductors between partial circuits. In turn, the circuit analyzer provides the circuit completer with information on current flow through the conductors between partial circuits.

The number of partial circuits and paths between them are defined when the model of the circuit of interest is developed. The solenoid-operative valve control circuit defined for this example is comprised of six partial circuits with three paths (1-2, 3-4, and 5-6) between each of two partial circuits.

To model the Circuit Completer Object, one state, Complete Circuit, is defined as indicated in the state transition diagram in Figure 5-25. No state transitions are defined for this object; however, once this object is initialized, it will progress to generate all possible conditions *and orders* of completion for the paths between the partial circuits. The attributes for this object are defined in terms of the paths between partial circuits in the circuit of interest. For the solenoid-operative valve control circuit, three paths are defined and thus the attributes are PATH 1-2, PATH 3-4, and PATH 5-6. Five values are defined to set the conditions for each path (Closed, Open, Short to +, Short to -, and Short to Each Other).

The conditions of completion for the paths between partial circuits are embedded in the DRM for the State:(Complete Circuit). No IRM is defined because the attributes for this object are not dependent on what happens in other objects. The initial conditions of completion are set for

normal operation (PATH 1-2=(Closed), PATH 3-4=(Closed), and PATH 5-6=(Closed)). Subsequently, the branching logic “randomly” generates a change in the overall model’s completion characteristics by changing one of the paths to be (Open), (Short to +), or (Short to -), or by shorting two or more of these paths to each other. The DRM then executes a prescribed delay, then “randomly” generates another change in these completion characteristics. The process is repeated until an analyst-specified time has expired (which will correspond to a pre-defined number of changes in the model’s completion characteristics). All combinations and orders of completion characteristics will eventually be explored because the OBEST solution algorithm conducts an exhaustive search for all possible scenarios. Thus, the “random” nature of the DRM completion logic does not detract from the completeness of the overall circuit analysis.

Figure 5-25. State transition diagram for the Circuit Completer Object.

The DRM model employed in the circuit completer object uses probabilities that embody the relative likelihood of each possible completion condition, and the delay time represents the time that elapses between changes in the circuit’s completion characteristics. If a scientific basis exists for either the relative probabilities or the delay time, this information can be incorporated directly into this object’s DRM. For this model, however, the probability values were set to be equal (based on the number of possible adverse conditions to be considered) because no such scientific basis was assumed. Thus, the *likelihood* of each path is not meaningful for this model – only the insights gained from the path definitions themselves. For the solenoid-operative valve control circuit, thirteen adverse conditions are considered, as indicated in Table 5-12, each with a probability of $1/13$. For nine conditions, each path is individually set to either (Open), (Short to +), or (Short to -). Additionally, four combinations of the shorts to each other are possible: (1) PATH 1-2 and PATH 3-4; (2) PATH 1-2 and PATH 5-6; (3) PATH 3-4 and PATH 5-6; and PATH 1-2, PATH 3-4 and PATH 5-6. After each change in the condition of completion is made, the circuit completer object generates the Event:Reset/New Conditions, which causes a state transition in the circuit analyzer object as described in the next section. The circuit analyzer then assesses the current flows within the system model and generates the appropriate values for the various partial circuit and component objects. The system then waits for the circuit completer’s DRM clock to expire, and the process repeats itself.

Table 5-12. Probabilities for adverse conditions of completion.

Attribute	Attribute Values for Adverse Conditions of Completion						
	Open	Short to +	Short to –	Short to Each Other (4 possible combinations)			
				1-2/3-4	1-2/5-6	3-4/5-6	All
PATH 1-2	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}^{(1)}$		$\frac{1}{13}$
PATH 3-4	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$				
PATH 5-6	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$		$\frac{1}{13}^{(1)}$	$\frac{1}{13}$	

⁽¹⁾ Two parts of a single short combination – Path 1-2 to Path 5-6

5.4.8. Circuit Analyzer Object

The Circuit Analyzer Object (CA) infers conditions of current flow in the overall circuit based on conditions in the partial circuits and the circuit completer. The circuit analyzer provides the circuit completer with information on current flow through the conductors between partial circuits. In turn, the circuit completer provides the circuit analyzer with information on physical connections and electric potential for the conductors between partial circuits. Additionally, the circuit analyzer provides information to the partial circuit regarding current flow through the partial circuit. In turn, the partial circuit provides to the circuit analyzer information on its physical connections and electric potential.

To infer the conditions of current flow in a circuit, two states (Initialize and Assess) are defined for the circuit analyzer, as illustrated in the state transition diagram in Figure 5-26. These states are related by two events, *Analyze* and *Assess* that cause transitions between the two states, as indicated by the directed arrows in Figure 5-26. The Event:*Analyze* is generated by the circuit analyzer and causes a transition from the State:(Initialize) to the State:(Assess). The Event:*Reset/New conditions* is generated by the circuit completer object and causes a transition from the State:(Assess) to the State:(Initialize).

Three basic attributes (CURRENT_n, MAXIMUM + CURRENT POTENTIAL, and MAXIMUM – CURRENT POTENTIAL) are defined for this object to provide information on the current flow for each partial circuit. The first attribute, CURRENT_n, defines the current conditions for each partial circuit in the circuit of interest based on the conditions of the partial circuit and the circuit completion for the related path. This attribute represents the results generated by the circuit analyzer object. Four values (Normal, None, Short Circuit, Unknown) are defined for this attribute. A separate attribute CURRENT_n is defined for each partial circuit in the circuit of interest. For the solenoid-operative valve control circuit, six partial circuits are defined and thus the attributes are CURRENT₁, CURRENT₂, CURRENT₃, CURRENT₄, CURRENT₅, and CURRENT₆. For each short circuit condition, the other two attributes,

MAXIMUM + CURRENT POTENTIAL, and MAXIMUM – CURRENT POTENTIAL, are also evaluated. Four values (Open, Direct, Through Load, Unknown) are defined for these two attributes.

Figure 5-26. State transition diagram for the Circuit Analyzer Object.

The conditions for setting the attributes for the Circuit Analyzer Object are embedded in the IRM for the State:(Initialize). The initial conditions for the circuit analyzer’s attributes are set to await information from the partial circuits and the circuit completer. Thus, all attribute values are set to (Unknown). When the Event:*Reset/new conditions* is generated by the circuit completer object, this causes the circuit analyzer object to transition to the State:(Initialize) where the attribute values for the circuit analyzer are reset to initial conditions. Once the attribute values are reset, the Event:*Analyze* is generated by the State:(Initialize). This causes the object to transition into the State:(Assess), where the actual determination of current values within the system is accomplished.

The State:(Assess) has a very complex IRM and no DRM. The IRM examines the conditions within all six partial circuits plus the circuit completer to determine the currents that are flowing through this connectivity configuration. There is no DRM because the analyzer must complete its work “immediately” and report back its results to the circuit for recording or further processing. It would be prohibitively complex to systematically identify all possible combinations of polarities, connectivity conditions, open circuits, and loads, and to build these combinations into the IRM. Instead, the IRM accomplishes its modeling through the use of more general rules that must be evaluated in stages through the concept of “generations.” Here we make use of the fact that when the OBEST solution algorithm detects that any IRM has changed something, it determines that equilibrium has not yet been achieved, so an additional evaluation of the system’s IRMs (an additional generation) is ordered. Thus, we can solve one portion of the circuit analysis problem in one generation, and, as long as the rule that is applied causes an attribute change or state transition, the IRM will be called again for an additional generation in which additional rules can be applied and additional portions of the solution completed.

The details of the circuit analyzer's rules can be found in Appendix D. However, a summary of those rules is presented in the following paragraphs. The circuit analyzer first considers those partial circuits that are not in the completion state "shorted together." These circuits are relatively straightforward to analyze. Rules applied here include:

- If the circuit completer shows this path as "open," then there is zero current flow through these partial circuits.
- If the circuit completer shows this path as "closed," and either of the partial circuits to which it is attached is an open partial circuit (or not attached to a power supply), then there is zero current flow through these partial circuits.
- If the circuit completer shows this path as "closed," and both of the partial circuits to which it is attached are of the same polarity, then there is zero current flow through these partial circuits.
- If the circuit completer shows this path as "closed," and the partial circuits to which it is attached are of opposite polarities, and neither partial circuit includes a load, then there is a short circuit through these partial circuits.
- If the circuit completer shows this path as "closed," and the partial circuits to which it is attached are of opposite polarities, and both partial circuits includes a load, then there is insufficient current flow through these partial circuits.
- If the circuit completer shows this path as "closed," and the partial circuits to which it is attached are of opposite polarities, and one partial circuit includes a load, then there is normal current flow through these partial circuits.

Similar types of rules are employed for situations when the circuit completer shows this path is connected directly to a positive or negative polarity current source. When two or more paths are shorted together, several generations may be required to determine the current flow through the combined circuit. The method that is used to compute these currents applies the following basic algorithm:

1. Determine which of the partial circuits has the potential to provide the greatest potential current to the short circuit from a positive polarity current source, and note the value of that potential current in the attribute `MAXIMUM + CURRENT POTENTIAL`. The greatest potential current occurs when there is a partial circuit that is attached to a positive polarity current source and has no load (there is a direct connection from the short to the positive current source). The second greatest potential current occurs when there is a partial circuit that is attached to a positive polarity current source through a load. Otherwise, there is no potential positive polarity current.
2. Apply the same logic to determine which of the partial circuits has the potential to provide the greatest potential current to the short circuit from a negative polarity current source, and note the value of that potential current in the attribute `MAXIMUM - CURRENT POTENTIAL`.

3. Determine the current through the various partial circuits attached to the short.

- If the MAXIMUM + CURRENT POTENTIAL is a “direct connection” and the MAXIMUM – CURRENT POTENTIAL is also a “direct connection,” then there is a short circuit involving two or more partial circuits. All other partial circuits have no current.
- If either MAXIMUM + CURRENT POTENTIAL or MAXIMUM – CURRENT POTENTIAL is a “none,” then none of the partial circuits supports sufficient current flow.
- If both the MAXIMUM + CURRENT POTENTIAL and the MAXIMUM – CURRENT POTENTIAL are “through a load,” then none of the partial circuits supports sufficient current flow.
- If either MAXIMUM + CURRENT POTENTIAL or MAXIMUM – CURRENT POTENTIAL is a “direct connection,” and the other current potential is “through a load,” then these partial circuits support a normal current flow, and all other partial circuits have no current.

The actual rules that implement this algorithm are very complex – much more complex than they would need to be if the OBEST software supported a more powerful IRM and DRM model description language. As each partial circuit’s current is set (as an attribute) by the circuit analyzer, the other objects respond to those attribute changes through their IRM logic. Once all partial circuits’ currents are set, the OBEST model reaches equilibrium and awaits the time when the circuit completer’s DRM generates the *Event:Reset/New conditions*, when the process repeats itself.

5.4.9. Other Objects

This example problem has focused on the methods that would be used to analyze the electric circuits themselves, and especially to determine the algorithms that would have to be used:

- To impose the circuit completion variations on the circuit object model that might be caused by a fire event (the circuit completer object), and
- To analyze the circuit itself to determine the currents that flow through all of its segments for each completion combination (the partial circuits and circuit analyzer objects).

Thus, the object model described in this section simply models the electrical portions of this analysis. However, in order for this model to answer useful questions for an analyst, one needs more information. Essentially, the analyst wants to know whether the various circuit completion sequences generated by this OBEST model would present any unsafe conditions for the actual system operations. “Unsafe” may be defined as the system performing in a manner different from what the operator expects, or the definition may be more specific, noting particular conditions that either must occur or must be avoided. The analyst must specify a way for the OBEST software to know whether a situation is safe or unsafe. In some situations, the analyst

may be able to specify those conditions through the binning conditions. In other situations, the analyst will need to build a special object that “monitors” the conditions of the various circuit objects and sets states or attributes to denote the categories of circuit conditions that are of interest to the analyst. In still other situations, the analyst may wish to build one object that models how the system would act in the absence of electrical faults, and use either binning or a monitoring object to capture differences between the proper and fire-altered system behavior. In any case, it is possible to obtain a wide array of information from the OBEST analysis that can be used to understand how the circuit and system might respond under the uncertainties of a fire scenario, and to provide insights that can be used to help make these circuits and systems more predictable and functional under these challenging conditions.

5.5. Basic Infrastructure Problem

During the term of this LDRD project, the world experienced the now infamous “Y2K” problem, where it was feared that computer failures would cause widespread outages in major infrastructures such as electric power generation and distribution, natural gas pipelines, and telecommunications. This infrastructure problem, while not strictly related to the original purpose of the LDRD project, illustrates how infrastructures can interact with one another and how one can make various trades among replacement energy sources to achieve an optimal strategy to prepare for potential infrastructure disruptions in energy supplies. The OBEST method will be used in a manner similar to the police station – electric power model. We will develop a model that simultaneously represents all of the various preparation options considered, as well as a range of commercial infrastructure disruption scenarios. The analyst can modify the probabilities and attribute values within the various object models to represent different preparedness options using the same object model.

5.5.1. Problem Description and System Model

The initial situation, in which the analyst (or the “family” for which this analysis is being performed) is assumed to live in a moderately rural area and be unprepared for long-duration infrastructure disruptions. The family receives electric power and natural gas from “continuous” commercial infrastructures (i.e., they are delivered continuously from a pipeline or other direct connection). Since they are provided by the infrastructure “on demand,” they are difficult to stockpile for use during outages. Water is derived from a well that is pumped using electric power. In order to survive in relative comfort, the family needs to maintain access to water, heat, light, and heat for cooking (food is not considered in this analysis because it can be readily stockpiled in large quantities for later use). Many families have available flashlights, batteries, candles and matches as their only preparations for electric power outages. The system structure diagram for this “unprepared” situation is found in Figure 5-27.

When preparing for an outage of commercial infrastructures, one quickly realizes that most (if not all) critical functions in a typical home are strongly dependent on electric power for at least part of their functionality. Thus, one must decide either to provide a replacement source of electric power that can function during infrastructure disruptions (such as a generator), or to implement each of these functions in a manner that is independent of electric power. The first contingency case assumes that one will prepare for such outages simply by providing a generator

(stocked with fuel) for use during times of severe infrastructure disruptions. Figure 5-28 shows the system structure diagram for this situation.

Figure 5-27. System structure diagram for the “Unprepared” situation.

The other major approach to prepare for major infrastructure disruptions is to implement the major household functions in a manner that is independent of electric power. One might provide heat for the house using firewood and/or a kerosene heater. Light can be provided by candles, flashlights, and/or a gas-powered camping lantern. Heat for cooking can be provided by a gas-powered camp stove and/or a charcoal or propane-fired barbecue grill. And drinkable water can be obtained by storing bottled water, or by using cooking heat to boil non-potable water that might be found or stockpiled outside in this rural environment. Figure 5-29 shows the system structure diagram for this “no electricity” scenario.

For this analysis, we wanted to model many combinations of preparedness methods in order to assess a variety of preparation options. For this reason, we combined all of the aspects of the “unprepared,” “with generator,” and “no electricity” options into the combined (or unified) object model shown in Figure 5-30. The object descriptions for this model are found in Table 5-13, while the interactions diagram can be found in Figure 5-31 (the interactions themselves are noted and described in Table 5-14). Using a combined model, an analyst can add or remove various preparation options from consideration with a simple change of an object’s initial condition (to reflect its presence or absence) and then quickly rerun the analysis to determine the effect of that change on system behavior.

Figure 5-28. System structure diagram for the “With Generator” situation.

Figure 5-29. System structure diagram for the “No Electricity” situation.

Figure 5-30. System structure diagram for the combined model.

Figure 5-31. Interactions diagram for the combined model.

Table 5-13. Objects in the combined structure diagram.

Object Name	Object Description
Water	Drinking/cooking water availability
Heat	Space heat availability
Cooking	Heat for cooking availability
Light	Minimal light availability
Electricity	Availability of electricity
Cm. Power	Availability of commercial electrical power
Bottles	Bottled water reserve
Com. Gas	Availability of commercial (pipeline) gas
Batteries	Battery reserve for flashlights
Candles	Candle reserve for lighting
Matches	Reserve of matches
Generator	Reserve electric generating capacity (includes fuel supply for generator)
Kerosene	Fuel reserve for kerosene heater
Firewood	Firewood reserve for heating
Camp Fuel	Fuel reserve for camp stove, lantern

Table 5-14. Functional interactions within the combined model.

Objects	Interactions
Com. Gas – Heat	Commercial Gas provides fuel to Heat
Gas – Cooking	Commercial Gas provides fuel to Cooking
Com. Power – Electricity	Commercial Power provides electric power to Electricity
Generator – Electricity	Generator provides electric power to Electricity (Demand for Electricity passed back)
Electricity – Light	Electricity provides power to provide Light (Demand for Electricity passed back)
Electricity – Cooking	Electricity provides power to enable Cooking (Demand for Electricity passed back)
Electricity – Heat	Electricity provides power to provide Heat (Demand for Electricity passed back)
Electricity – Water	Electricity provides power to pump Water from the well (Demand for Electricity passed back)
Batteries – Light	Batteries provide power to provide Light (Demand for Batteries resources passed back)
Candles – Light	Candles provide energy to provide Light (Demand for Candles resources passed back)
Matches – Light	Matches provide fire that enables candles or lanterns to provide Light (Demand for Matches resources passed back)
Matches – Cooking	Matches provide fire that enables camp stove to provide Cooking (Demand for Matches resources passed back)
Matches – Heat	Matches provide fire that enables firewood or kerosene heater to provide Heat (Demand for Matches resources passed back)
Cooking – Water	Cooking provides boiling to purify Water
Bottles – Water	Bottles provide reserves to Water (Demand for water resources passed back)
Kerosene – Heat	Kerosene provides fuel to Heat (Demand for Kerosene resources passed back)
Firewood – Heat	Firewood provides fuel to Heat (Demand for Firewood resources passed back)
Water – Camp Fuel	Water provides an (additional) energy demand to Camp Fuel
Camp Fuel – Light	Camp Fuel provides energy to provide Light (Demand for Camp Fuel is passed back)
Camp Fuel – Cooking	Camp Fuel provides energy to provide Cooking (Demand for Camp Fuel is passed back)

5.5.2. Individual Object Models

Each element in the system structure and system interaction diagrams presented in the previous section (especially Figure 5-30 and Figure 5-31) is represented as an object in the OBEST model of this system and its behaviors. Thus, this OBEST model is composed of 15 separate objects. However, by breaking the system into so many objects, we are able to keep the definition of each object extremely simple. No object in the system has more than two states, and no IRM or DRM in the system contains more than six cases. This section is dedicated to the description of these 15 objects and their interactions.

The first four objects to be discussed are the four vital household functions that are to be maintained during infrastructure disruptions: the Light Object, the Heat Object, the Cooking Object, and the Water Object. These objects are similar to one another in that each functions to track the various sources that can provide its vital function and ultimately determines whether or not that function is being met. Each has two states: one where the function is being met, and one where the function has failed. The transition between these states occurs when the energy supply on which the object relies becomes depleted or otherwise unavailable (the event that causes this state transition is generated within this object's own IRM). Each of these objects has one attribute that represents the source from which that vital function is drawing its energy. Since these objects track the availability of these vital functions (and do not themselves provide the energy upon which these functions rely), none of these objects has a DRM. Furthermore, since this analysis does not seek to model the recovery of these vital functions, there is neither functionality contained in nor a transition out of the state in which the vital function is not being provided. The IRM for the functional state simply searches other objects to determine whether any of them are capable of providing energy to support this vital function.

The state transition diagram for the Light Object is found in Figure 5-32. There is no functionality in State: (Dark), and the functionality in State: (Light), which is embodied only in the IRM, determines whether the attribute that represents the source of power for light is set to electric power, lantern, flashlight, candles, or none. Note that lantern and candles can only be used if the supply of matches has yet to be exhausted. If none of these energy sources remains available to support the Light Object's vital function, this IRM "turns out the lights" by generating the Lights Out event that causes this object to transition to state: (Dark).

The state transition diagram for the Heat Object is found in Figure 5-33. There is no functionality in State: (No Heat), and the functionality in State: (Warm) determines whether the attribute that represents the source of energy for heat is set to the regular furnace, a kerosene heater, a fireplace, or none. This functionality is embodied in the IRM, which will not allow either the kerosene heater or the fireplace to be used unless the supply of matches is not yet exhausted. If none of these energy sources remains available to support the Heat Object's vital function, this IRM generates the Heat Out event that causes this object to transition to State: (No Heat).

The state transition diagram for the Cooking Object is found in Figure 5-34. As with the object models for the other vital functions, there is no functionality in State: (No Cooking), and the functionality in State: (Cooking) uses the IRM to determine whether the source of energy for

cooking is the regular stove, a camping stove, or none. The attribute that represents this source is set to the appropriate value within the IRM. This IRM will not allow the camping stove to be used if the supply of matches has already been exhausted. If none of the energy sources that support the Cooking Object's vital function remains available, this IRM generates the Cooking Out event that causes this object to transition into State: (No Cooking).

Figure 5-32. State transition diagram for the Light Object.

Figure 5-33. State transition diagram for the Heat Object.

Figure 5-34. State transition diagram for the Cooking Object.

The fourth and final vital function object is the Water Object found in Figure 5-35. All of the functionality for this object is incorporated in the IRM in State: (Consuming), which sets an attribute to indicate whether the source of potable water for the home is the well, bottled water, boiled water, or none. Note that boiled water cannot be produced if the cooking vital function is unavailable. Should all sources of water become unavailable, this IRM generates the Water Out

event that causes this object to transition to State: (No Water). The State: (No Water) embodies no functionality of its own.

Figure 5-35. State transition diagram for the Water Object.

The Electricity Object represents a variation on the theme that was generated for the vital functions objects. Its purpose is to summarize the present source and status of electric power for the other entities in the object model. This functionality is implemented in a single state with an IRM and no DRM, as seen in Figure 5-36. The use of a single state for the Electricity Object (as opposed to two states for the vital functions objects) means that the Electricity Object will always represent the actual current state of electric power in the system, while the various vital functions objects will “latch” in their nonfunctional states should they ever occur. This latching behavior helps the analyst to determine whether a vital function was lost *at any time* during the scenario. Since electric power is not itself a vital function, the latching behavior was omitted from the Electricity Object.

Figure 5-36. State transition diagram for the Electricity Object.

The Electricity Object uses one attribute to indicate whether electric power is available or unavailable, and a second attribute to identify the source of that electric power (commercial electric power, a generator, or none). The sole function of this object’s IRM, then, is to set appropriate values for these attributes based on the characteristics of the other objects in the system.

Eight of the remaining 10 objects in the infrastructure disruption preparation model represent consumable supplies that would be used in the event of a major infrastructure outage. These objects are all extremely similar in their structure in behavior. Each has one attribute named Supply that indicates whether the reserve supply of this commodity remains available or has been exhausted. These two discrete values for the attribute correspond to the two states that

make up each of these object models – again, one to indicate that this commodity remains available and one to indicate that it is unavailable. Each object’s IRM is used to ensure the proper setting of the Supply attribute and to manipulate the object’s internal clock. The clock represents a sort of “countdown” until this object’s resources will be exhausted. The delay that is built into each object’s DRM indicates the number of days the analyst expects this resource to last when used in a particular fashion. The object’s clock is started when the appropriate vital functions object indicates that this object’s supplies are being used as a source to fulfill that vital function’s needs. When the delay time is exhausted, this object’s resources are assumed to be fully depleted, so the DRM generates an event that will cause the object to transition into its depleted or unavailable state. The DRM may include multiple delay times to indicate that different usage rates occur when the resource is used in different ways. For example, camp fuel may last 15 days if it is used only for lighting, seven days if it is used for lighting and cooking, and only four days if one is relying on boiled water. The object selects the appropriate delay by reading the attributes from the various vital functions objects and comparing those attributes to the logical cases in the object’s DRM.

The state transition diagram for the Camp Fuel Object is found in Figure 5-37. The object’s states are Fuel and Empty, and the corresponding attribute values are Fuel and None. In State: (Empty), the object sets the attribute value to None and stops the object’s internal clock to indicate that there is no functionality embodied in this state. In State: (Fuel), the object’s IRM ensures that the attribute value is set to Fuel and enables or disables the object’s clock as dictated by fuel demands from the Cooking Object and the Light Object. The DRM implements the delay times described in the previous paragraph to reflect the different lengths of time the supply of camp fuel would be expected to last under different demand conditions.

Figure 5-37. State transition diagram for the Camp Fuel Object.

The Bottles Object implements a similar functionality for a supply of bottled potable water that a person may have stockpiled as a hedge against a protracted infrastructure outage. The state transition diagram for this object, as shown in Figure 5-38, uses the states Water and Empty with the corresponding attribute values Water and None. The functionality embodied in State: (Empty) is identical to that embodied in State: (Empty) in the Camp Fuel Object. In State: (Water), the IRM ensures that the attribute value is set to Water and enables or disables the object’s clock depending on the value of the Source attribute in the Water Object. The DRM in this object implements a five-day delay to model the fact that the analyst believes that the stockpile of bottled water will last for five days under the expected usage conditions.

Figure 5-38. State transition diagram for the Bottles Object.

The Batteries Object represents the supply of batteries that one might have on hand to power flashlights for use as a light source during an infrastructure disruption. The state transition diagram for this object is shown in Figure 5-39. It uses the states Power and Dead with the corresponding Supply attribute values of Batteries and None. State: (Dead) embodies the same functionality as found in the Empty states of the previous two objects. In State: (Power), the IRM ensures that the attribute value is set to Batteries and that the object's clock is enabled or disabled according to the demands placed on this power source by the Light Object. The seven-day delay implemented in this object's DRM represents the analyst's belief that the proposed reserve supply of batteries would enable a flashlight to provide light for seven days.

Figure 5-39. State transition diagram for the Batteries Object.

The Candles Object models the reserve supply of a second source of light that might be used in the event of electric power failure: candles. Note that this object represents only the supply of candles themselves, and not the matches that would be required to light them. As seen in Figure 5-40, this object uses the states Candles and Gone with the corresponding attribute values Candles and None. The State: (Gone) implements the same functionality as the Empty states in the first two consumables objects. State: (Candles) uses an IRM to ensure that the attribute value is set to Candles and to enable or disable this object's clock as required by the Light Object. A three-day delay is implemented in the DRM to represent the three-day supply of candles that this home might have on hand.

The supply of fuel for a kerosene heater is represented by the Kerosene Object. The state transition diagram shown in Figure 5-41 makes use of the states Kerosene and Empty and the

corresponding Supply attribute values of Kerosene and None. State: (Empty) implements the same functionality as the Empty states in the first two consumables objects. State: (Kerosene) uses its IRM to enable or disable this object's clock in accordance with the demands levied by the Heat Object, as well as ensuring that the Supply attribute is set to the appropriate value. The DRM in this state implements a five-day delay to account for the expected usage rate and the supply of kerosene presumed to be on hand.

Figure 5-40. State transition diagram for the Candles Object.

Figure 5-41. State transition diagram for the Kerosene Object.

The OBEST infrastructure disruption preparation model assumes that the home for which preparations are being made has a fireplace or wood stove that can be fueled for at least ten days with the supply of firewood that is on hand. This functionality is modeled in the Firewood Object, whose state transition diagram is shown in Figure 5-42. The Firewood Object makes use of the states Firewood and Gone, as well as the corresponding attribute values Firewood and None. The IRM and DRM in State: (Gone) are the same as those found in the Empty states of the first two objects. The functionality implemented in the IRM and DRM for State: (Firewood) is the same as that found in the Kerosene Object, with the exception that the DRM delay represents a ten-day supply of firewood as opposed to the five-day supply of kerosene.

Several of the fuel supplies that are modeled in the previously mentioned objects depend on the use of matches to start the fire that will enable the vital function to be performed. The Matches Object models the supply of matches that is available for this purpose. This object makes use of two states, Matches and Gone (as shown in Figure 5-43), and two corresponding values for the Supply attribute: Matches and None. As with previous object models for consumable materials,

State: (Gone) sets the object's attribute value to None and turns off the object's clock to indicate that there is no functionality embodied in this state. State: (Matches) implements a delay of 12 days to model the 12-day supply of matches that the analyst believes will exist when infrastructure disruption occurs. The IRM in this state simply ensures that the Supply attribute is set to the appropriate value and enables or disables the object's clock in accordance with the demands for matches by the Heat Object, the Cooking Object and the Light Object.

Figure 5-42. State transition diagram for the Firewood Object.

Figure 5-43. State transition diagram for the Matches Object.

The final “consumables” object embodied in this OBEST model is the Generator Object. As currently configured, this assessment assumes that the generator will produce electricity as long as it has fuel. That is, this model does not consider failures in the generator equipment itself that might render it incapable of producing electricity even though it still has remaining fuel. Generator failure models could be incorporated in this object, but they were considered beyond the scope of this analysis. The state transition diagram for the Generator Object is shown in Figure 5-44. It consists of the states Available and Dead, and the object's state is reflected by the value of the Supply attribute, which can take on the discrete values of Available and None. The behavior of this object in State: (Dead) mirrors the behavior of the other consumables object models. Its behavior in State: (Available) is also similar to that seen for the other consumables object models in that the IRM ensures that the Supply attribute is set to an appropriate value, and starts and stops the object's clock as appropriate based on demand from the Electricity Object. The DRM in this state implements a nine-day delay to embody the nine-day supply of fuel that has been proposed as an infrastructure disruption preparedness measure.

Figure 5-44. State transition diagram for the Generator Object.

The remaining two objects in this OBEST model represent the availability of commercial electric power and commercial natural gas. These objects are probabilistic in nature, and are designed to provide the remainder of the OBEST model with a variety of “boundary conditions” that represent the array of possible infrastructure outages that might have been expected as a result of the transition to the year 2000. Each object has two states: Available and Dead, and one attribute named Supply. The Supply attribute can take on three values: “Available” if the infrastructure is available because it has yet to be exposed to the potential for failure (i.e., the transition to the year 2000 has not yet occurred); “Restored” if the infrastructure is available because it has already been exposed to the potential for failure and either did not fail or has been repaired; and “None” if this infrastructure is not available to perform its designed function. The state transition diagram for the Commercial Power Object is shown in Figure 5-45, while the diagram for the Commercial Gas Object is shown in Figure 5-46.

The commercial infrastructure objects function as follows. Initially, each object is in State: (Available) with the Supply attribute set to the Available value. State: (Available) has no DRM, and its IRM is used to determine whether this infrastructure experiences an outage condition or it sails right through the transition to the year 2000 without any disruption. This decision is implemented as a probabilistic branching operation, with a likelihood of 10 percent for no disruption and 90 percent for some kind of disruption. If the supply is not disrupted, the attribute value is changed from Available to Restored to indicate that this infrastructure has been exposed to the potential for failure but did not actually fail. However, if disruption occurs, this IRM generates an event to cause the object to transition to State: (Dead). The only function of the IRM for State: (Dead) is to reset the value of the Supply attribute from Available to None when the object enters this state. The DRM in this state will be used to probabilistically determine the length of the infrastructure outage. For this screening analysis, we have assigned identical likelihoods of 10 percent to each of 10 possible outage durations (delay times) that range from a minimum of 0.5 days to a maximum of 40 days. At the end of the outage (after the delay time has expired), the DRM generates an event to cause the object to transition back into State: (Available), where the IRM will change the value of the Supply attribute from None to Restored to indicate that this outage is over and the infrastructure is once again performing its intended function.

Figure 5-45. State transition diagram for the Commercial Power Object.

Figure 5-46. State transition diagram for the Commercial Gas Object.

Now that we have described the objects that are involved in this infrastructure disruption preparation problem, let us take a moment to consider how all of these pieces fit together into one overall system model. Initially, all four of the vital functions are supplied with energy from either the Commercial Power Object or the Commercial Gas Object. As the transition to the year 2000 occurs, one or both of these objects might move into State: (Dead) to indicate an outage has occurred. The Electricity Object responds to this situation by calling for electric power from the Generator Object. If there is no generator available (or when fuel for the generator runs out), the four vital functions objects look to their other potential sources to determine whether the vital functions continue to be provided. Each vital function goes through its potential sources of energy in an order of preference. For example, the Heat Object first looks to kerosene, then to fire wood, while the Light Object first looks to a camping lantern, then to flashlights, then to candles. If any source is unavailable – either because its supply is exhausted or because its capacity was set to zero for this analysis (to indicate, for example, that the owner did not wish to purchase a portable generator) – then the vital function object moves on to its next preferred source of energy. If the vital function object cannot find an appropriate source of energy, then it enters into an “unavailable” state that denotes that, at some point during the analysis, this vital function could not be performed. Eventually, as the delay times associated with the commercial infrastructure objects expire, commercial power and commercial gas supplies are restored. Each scenario is terminated when both commercial power and commercial gas are restored. At that point, one can examine the four vital functions objects which, because of their latching behavior,

will remain in their “unavailable” states if there was any time during the scenario during which their function could not be performed.

The ultimate result to be obtained from running this model is a series of scenarios that will enable the analyst to determine the scenarios that might lead to failure for one or more of these four vital functions. The probabilistic weights for those scenarios are determined based on the probabilistic branching embodied in the commercial infrastructure objects, and ultimately may embody the analyst’s expectations regarding the likelihood of various durations of infrastructure outages that might occur as a result of the then still future transition to the year 2000. The analyst could run this model under a variety of sets of assumptions related to outage durations (changing the delay times in the commercial infrastructure objects DRMs) and personal preparations. Changes in personal preparations might reflect the inclusion or exclusion of particular energy sources (implemented by changing the initial attribute values and states for particular objects). These changes might also reflect options for purchasing different reserve quantities for particular energy sources (implemented by changing the DRM delay values for the appropriate consumables objects). These analyses, when taken together, would provide the analyst with a valuable picture of how his or her current expectations would be expected to play out when the actual transition to year 2000 occurred.

5.5.3. Summary and Insights

While this example problem did not directly address the “infrastructure interdependencies” aspect of the original LDRD proposal, it should be easy to see how the concepts developed in this example problem can be applied to these interdependency problems. Indeed, this problem was selected because it revealed most of the same issues that would be examined in an infrastructure interdependencies problem, but did so in a format that would be easy for the uninitiated reader to identify with. This problem demonstrated how the failure of one infrastructure (commercial electric power, for example) can impact a number of vital functions simultaneously. It also demonstrated how a variety of remedies may be available to deal with infrastructure outage situations, and how those various remedies can be embodied in a single object model and exercised in various combinations without the need to revise the object model itself. The principal element that was missing from this model but might be present in a more generalized infrastructure interdependencies problem was feedback between infrastructures. However, feedback was demonstrated in the “Electric Power for a Police Station” example problem in Section 5.1. Thus, between these two sample problems, we have covered all of the major aspects that might be involved in a real infrastructure interdependencies problem.

As we worked this infrastructure disruption problem, it became clear that one could solve a relatively complex problem by building many objects – each of which, by itself, is quite simple – and allowing them to interact with each other in such a way that we can observe the emergent behavior. The OBEST model provides a mechanism for generating this behavior with capabilities far beyond traditional object based techniques and agent based simulation methods in that it can generate a whole family of probabilistically weighted scenarios that are self consistent, and do so rapidly. It is this generation of an entire family of solutions that can be examined *together in context* that leads to the system-wide insights that have been so beneficial from risk assessment studies.

The infrastructure disruption model that was developed and described in this chapter can be expanded in a number of ways. One might, for example, incorporate probabilistic branching into the various consumables object models to represent the uncertainty that exists in that commodity's demand characteristics. One might also incorporate probabilistic branching to model mechanical problems in the equipment being used (e.g., failure of the generator to start, contamination in "old" can fuel, etc.). One might also introduce gradations into the four vital functions objects to represent the fact that, for example, the light produced by candles is of a lower intensity and, thus, less useful than that produced by electric lights or by a camping lantern. In this way, one might include "quality of life" issues in scoring the relative merits of potential infrastructure preparedness remedies.

The methods used to develop this model do point out one major limitation in the OBEST methodology and OBEST software as it currently exists. To generate each consumables object, we were required to combine the commodity's expected usage rate with the amount of the commodity in on hand storage to determine an estimate for the time that would elapse before the stored commodity would be exhausted. We relied on the concepts of "nonlinearity" in DRM time, as described in Section 3.3, to establish transitions in usage rates for commodities such as camp fuel when additional demands were placed on the commodity to cook or boil potable water. We believe it would be much more accurate and natural to represent the storage capacity and utilization rate for each commodity as a separate value that would be combined within the OBEST software to determine the time at which each DRM would expire. It is our hope that this capability can be implemented in the OBEST software soon.

5.6. Aircraft Spacing During Runway Crossing Operations

Aviation safety is another area of application for which the OBEST methodology is being investigated. Currently, Sandia is providing support in the area of risk and reliability analysis for the joint National Aeronautics and Space Administration (NASA) and Federal Aviation Administration (FAA) Aviation Safety Program. One area of emphasis for the Aviation Safety Program is the development of advanced techniques for modeling safety issues in the national airspace. Sandia has proposed using OBEST as a effective modeling method for developing system safety models that integrate human performance as well as the operations of and relationships among the numerous components of the complex aviation system environment.

Runway operations is an important area of concern in aviation safety. As part of this LDRD project, we developed a preliminary model for this example problem that has possible application for a number of runway operations. These models considered aircraft spacing for runway crossings, as well as risk assessment for runway collisions, runway incursions and for land-and-hold-short operations.

The OBEST system model for this application included components to model the airport/airspace, a landing aircraft, a taxi aircraft, the air traffic controller, and the environment (for example, weather, radio conditions, traffic levels). Due to resource constraints, only the airport/airspace, landing aircraft, and taxi aircraft components were modeled in detail.

5.6.1. System Structure and Interactions

The system structure diagram (Figure 5-47) for the model is defined by five functional blocks. The interaction diagram is shown in Figure 5-48, and the functional interactions for the model are specified in Table 5-15. Interactions occur among the functional blocks, as indicated in Figure 5-48 and Table 5-15. Each of the functional blocks is described in the following paragraphs. The descriptions of the Airport/Airspace, Taxi Aircraft and Landing Aircraft objects include a discussion of their respective state transition diagrams. The discussion of the state transition diagrams includes a description of the object, events, and event transitions, and a narrative of the “if – then – else” scripts and logical conditions for the IRM and DRM for each state. The detailed “if – then – else” scripts and logical conditions are provided in Appendix D.

Figure 5-47. System structure diagram for model of runway operations.

Figure 5-48. Interaction diagram for model of runway operations.

Table 5-15. Functional interactions for model of runway operations.

AP-TC:	The Airport/Airspace provides to the Tower Controller information on aircraft positions.
TC-AP:	The Tower Controller provides to the Airport/Airspace information about runway reservations.
TC-TA:	The Tower Controller provides taxi commands to a Taxi Aircraft and responds to requests from a Taxi Aircraft.
TA-AP:	A Taxi Aircraft provides to the Airport/Airspace information about its position.
TA-TC:	A Taxi Aircraft makes requests to the Tower Controller and responds to commands from the Tower Controller.
LA-AP:	A Landing Aircraft provides to the Airport/Airspace information about its position.
Env-TC:	The Environment provides to the Tower Controller information about distractions, visibility, and radio interference.
Env-TA:	The Environment provides to a Taxi Aircraft information about distractions, visibility, and radio interference.
Env-LA:	The Environment provides to a Landing Aircraft information about distractions, visibility, and radio interference.

5.6.2. Airport/Airspace Object

The Airport/Airspace Object provides a physical layout of the runways and taxiways and the airspace of an airport. For this model, a description of the physical layout of one runway and one taxiway and their intersection was developed. This physical layout is presented in Figure 5-49. The runway and taxiway are divided into segments, each of which represents discrete values for locations to be occupied by aircraft. For purposes of simplification, runway segments R2 and R3 are merged (R23), and taxiway segments T1 and T2 are merged (T12). Aircraft separation is defined in terms of the locations of a taxi aircraft and a landing aircraft. The potential for a collision or a runway incursion can be evaluated by comparing the runway and taxiway segments occupied by a set of aircraft. Table 5-16 provides a listing of the combinations of runway and taxiway locations, the corresponding aircraft separation, and a severity ranking of that combination of locations. The severity rankings for the different combinations of locations are arbitrary and were defined by project staff for the purposes of this example problem.

The Airport/Airspace Object represents the physical layout of the airport and tracks the locations of the Taxi Aircraft and Landing Aircraft within this layout. The Taxi Aircraft and Landing Aircraft objects provide to the Airport/Airspace Object information about their respective locations. The Airport/Airspace Object, in turn, provides this location information to the Tower Controller.

To model the Airport/Airspace Object, one state, Airport Status, is defined as indicated in the state transition diagram in Figure 5-50. No state transitions are defined for this object because the behavior of physical layout of the Airport/Airspace does not change. Changes in airport and airspace conditions are modeled through the use of attributes. Three attributes are defined for this object to provide information on aircraft locations and runway status. The first attribute, Taxi Aircraft Location – TAL, defines the location of the Taxi Aircraft. Six values (G, T12, T3, X, T5, BT) are defined for this attribute. For initial conditions, the Taxi Aircraft is at the gate (TAL=(G)). The second attribute, Landing Aircraft Location – LAL, defines the location of the Landing Aircraft. Six values (A, R1, R23, X, R5, BR) are defined for this attribute. For initial conditions, the Landing Aircraft is airborne (LAL=(A)). The third attribute, Runway Status – RS, defines the runway status. Two values (Reserved, Available) are defined for this attribute. For initial conditions, the runway is reserved (RS=(Reserved)).

The State:(Airport Status) tracks the locations of the Taxi Aircraft and Landing Aircraft within the physical layout of the Airport/Airspace and the status of the runway. The Airport/Airspace Object considers the location information from the Taxi Aircraft and Landing Aircraft, and as these locations change, updates the Airport/Airspace location attributes to correspond with this information. For the runway status, if the Tower Controller clears a Landing Aircraft to land, then the runway status is set to reserved (RS=(Reserved)). If the aircraft is not cleared to land, then the runway status is set to available (RS=(Available)). The conditions for this state are embedded in the IRM. No DRM is defined.

Figure 5-49. Physical layout for the Airport/Airspace Object.

Figure 5-50. State transition diagram for the Airport/Airspace Object.

Table 5-16. Collision potential for combinations of runway and taxiway locations.

Runway Location	Taxiway Location	Aircraft Separation	Severity Rank
X	X	Collision	1
R23	X	Very Close Call 1	2
X	T3 (Not stopping)	Very Close Call 2	3
X	T5	Very Close Call 3	4
R5	X	Very Close Call 4	5
R1	X	Close Call 1	6
BR	X	Close Call 2	7
A	X	Incursion	8
All Other	All Other	No Problem	9

5.6.3. Landing Aircraft Object

The Landing Aircraft Object (LA) represents the behavior of a landing aircraft. As defined for this model, the Landing Aircraft provides to the Airport/Airspace Object information about its respective locations. The Environment would provide to the Landing Aircraft information on environmental conditions including weather, visibility, radio interference, air traffic, and other environmental distractions. To model the Landing Aircraft Object, four states are defined as illustrated in the state transition diagram in Figure 5-51. These states include Change Location, Flying, Moving, and Stopped. These states are related by a set of events that represents the Landing Aircraft's behavior. These events cause transitions among the four states of the Landing Aircraft Object, as indicated by the directed arrows in Figure 5-51. The set of events defined for the model includes: *LA_New_Location (LA-NL)*, *LA-Fly*, *LA-Move*, and *LA-Stop*.

Three attributes are defined for this object to provide information about the Landing Aircraft's behavior and location. The first attribute, Mode, defines the mode of operation for the Landing Aircraft. Two values (Takeoff, Landing) are defined for this attribute. For initial conditions, the Landing Aircraft is flying in for landing (Mode=(Landing)). The second attribute, Location - LAL, defines the location of the Landing Aircraft. Six values (A, R1, R23, X, R5, BR) are defined for this attribute. For initial conditions, the Landing Aircraft is airborne (LAL=(A)). The third attribute, Motion, defines the Landing Aircraft's motion. Three values (Stopped, Moving, Flying) are defined for this attribute. For initial conditions, the Landing Aircraft is flying in for landing (Motion=(Flying)).

Figure 5-51. State transition diagram for the Landing Aircraft Object.

The State:(Flying) is defined to set the location of the Landing Aircraft as it begins to fly in for landing. First, the object attributes are set for the Landing Aircraft in this state (Location=(A), Motion=(Flying)). These conditions are embedded in the IRM. Then, after combinations of probabilities and specified delay times, the Event:*LA_New_Location* is generated and the Landing Aircraft proceeds to a new location. These conditions reflect a probabilistic timing that the aircraft will occupy different segments of the runway. Three sets of timing conditions are defined for the model: P=0.25 for Delay Time=120; P=0.50 and Delay Time=135; and P=0.25 for Delay Time=150. These conditions are embedded in the DRM.

The State:(Change Location), moves the Landing Aircraft down the runway from segment to segment. When the Mode=(Landing) and the Location=(A), this state will move the aircraft from segment A to segment R1 in response to the Event:*LA_New_Location*. The aircraft will continue to move from segment R1 to R23 to X to R5 as the Event:*LA-Move* is successively generated. When the aircraft location is in segment BR, then the object attributes are set to Motion=(Stopped) and the Event:*LA-Stop* is generated. The state is also used for an aircraft taking off as it proceeds from segment BR to segment A with the successive generation of the Event:*LA-Move*. Then the object attributes are set to Motion=(Flying) and the Event:*LA-Fly* is generated. The conditions for this state are embedded in the IRM. No DRM is defined.

The IRM for the State:(Moving) sets the Landing Aircraft Motion=(Moving). The DRM sets the probabilistic timing for the aircraft motion down the runway as it moves from segment R1 to segment R5 when the aircraft is landing, or the reverse, R5 to R1, when the aircraft is taking off. This state does not apply for an aircraft in segments A or BR, because the respective motions for the aircraft in these locations are Motion=(Flying) and Motion=(Stopped). The probabilistic timing defined for aircraft movement in this state is provided in Table 5-17.

Table 5-17. Probabilistic timing for aircraft movement in State:(Moving).

Mode	Movement	Probability	Delay Time
Landing	R1 to R23	0.25	7
		0.50	10
		0.25	13
Landing	R23 to X	0.25	4
		0.50	5
		0.25	7
Landing	X to R5	1.00	3
Landing	R5 to BR	0.25	10
		0.50	15
		0.25	20
Takeoff	R5 to X	0.25	10
		0.50	20
		0.25	30
Takeoff	X	1.00	3
Takeoff	X to R23	0.25	5
		0.50	7
		0.25	9
Takeoff	R23 to R1	0.25	10
		0.50	13
		0.25	16

The State:(Stopped) is defined to set the location of the Landing Aircraft as it completes its landing or begins takeoff. First, the object attributes are set for the Landing Aircraft in this state (Location=(BR), Motion=(Stopped)). These conditions are embedded in the IRM. Then, after combinations of probabilities and specified delay times, the Event:*LA_New_Location* is generated, and the Landing Aircraft proceeds to a new location. These conditions reflect a probabilistic timing of the delay before the aircraft begins takeoff roll. Three sets of timing conditions are defined for the model: P=0.25 for Delay Time=15; P=0.50 and Delay Time=25; and P=0.25 for Delay Time=35. These conditions are embedded in the DRM.

5.6.4. Taxi Aircraft Object

The Taxi Aircraft Object (TA) represents the behavior of an aircraft on a taxiway. As defined for this model, the Taxi Aircraft provides to the Airport/Airspace Object information about its respective locations. The Environment would provide to the Taxi Aircraft information on environmental conditions including weather, visibility, radio interference, air traffic, and other environmental distractions. To model the Taxi Aircraft Object, five states are defined as illustrated in the state transition diagram in Figure 5-52. These states include Change Location, Gate Stop, Taxi, Hold, and Beyond Model. These states are related by a set of events that represent the taxi aircraft's behavior. These events cause transitions among the five states of the Landing Aircraft Object, as indicated by the directed arrows in Figure 5-52. The set of events defined for the model includes: *TA_New_Location (TA-NL)*, *TA-Gate*, *TA-Taxi*, *TA-Hold*, and *TA-Beyond Model*.

Figure 5-52. State transition diagram for the Taxi Aircraft Object.

Four attributes are defined for this object to provide information about the taxi aircraft's behavior and location. The first attribute, Mode, defines the mode of operation for the Taxi Aircraft. Two values (Takeoff, Landing) are defined for this attribute. For initial conditions, the Taxi Aircraft is preparing for takeoff (Mode=(Takeoff)). The second attribute, Location - TAL, defines the location of the Taxi Aircraft. Six values (G, T12, T3, X, T5, BT) are defined for this attribute. For initial conditions, the Taxi Aircraft is at the gate (LAL=(G)). The third attribute, Motion, defines the Taxi Aircraft's motion. Three values (Gate, Taxi, Hold, Beyond) are defined for this attribute. For initial conditions, the Taxi Aircraft is at the gate (Motion=(Gate)). The fourth attribute, PilotHold, is used to determine whether the Taxi Aircraft will hold position as directed by the Tower Controller. Three values (Unset, WillHold, NoHold) are defined for this attribute. For initial conditions, this attribute is unset (PilotHold=Unset).

The State:(Gate Stop) describes the Taxi Aircraft's behavior at the gate. First the attributes for this state are set (Location=(G), Motion=(Gate)). Then, if the Taxi Aircraft is preparing for takeoff (Mode=Takeoff), the probability that the Taxi Aircraft will comply with the direction of the Tower Controller (PilotHold=(WillHold)) is $P=0.999$.^{*} The probability that the Taxi Aircraft will fail to comply (PilotHold=(NoHold)) is $P=0.001$. These conditions are embedded in the IRM for this state. The DRM first considers if the Taxi Aircraft is landing. If it is and it is now located at the gate, then the scenario is terminated. Next the DRM considers delays leaving the gate. For different combinations of probabilities and specified delay times, the Event:*TA_New_Location* is generated and the Taxi Aircraft proceeds to a new location. Three sets of timing conditions are defined for the model: $P=0.25$ for Delay Time=100; $P=0.50$ and Delay Time=120; and $P=0.25$ for Delay Time=140. These conditions are embedded in the DRM.

The State:(Change Location), moves the Taxi Aircraft down the taxiway from segment to segment. When the Mode=(Takeoff) and the Location=(G), this state will move the aircraft from segment G to segment T12 in response to the Event:*TA_New_Location*. The aircraft will continue to move from segment T12 to T3 to X to T5 to BT as the Event:*TA-Taxi* is successively generated. When the aircraft location is in segment BT, then the object attributes are set to Motion=(Beyond) and the Event:*TA-Beyond Model* is generated. The state is also used for an aircraft landing as it proceeds from segment BT to segment G with the successive generation of the Event:*TA-Taxi*. Then the object attributes are set to Motion=(Gate) and the Event:*TA-Gate* is generated. The conditions for this state are embedded in the IRM. No DRM is defined.

The IRM for the State:(Taxi) sets the Taxi Aircraft Motion=(Taxi). The DRM sets the probabilistic timing for the aircraft motion down the taxiway as it moves from segment T12 to segment T5 when the aircraft is landing, or the reverse, T5 to T12, when the aircraft is taking off. This state does not apply for an aircraft in segments G or BT, because the respective motions for the aircraft in these locations are Motion=(Gate) and Motion=(Beyond). The probabilistic timing defined for aircraft movement in this state is provided in Table 5-18.

^{*} These probabilities and delays were estimated based on the judgement of the analysts without the aid of a formal human reliability analysis.

Table 5-18. Probabilistic timing for aircraft movement in State:(Taxi).

Mode	Movement	Probability	Delay Time
Takeoff	T12 to T3	0.25	30
		0.50	40
		0.25	50
Takeoff	T3 to X & PilotHold=(NoHold)	0.25	10
		0.50	15
		0.25	20
Takeoff	T3 to X & PilotHold=(WillHold)	0.25	10
		0.50	15
		0.25	20
Takeoff	X to T5	1.00	10
Takeoff	T5 to BT	0.25	15
		0.50	23
		0.25	30
Landing	T5 to X & PilotHold=(NoHold)	0.25	15
		0.50	23
		0.25	30
Landing	T5 to X & PilotHold=(WillHold)	0.25	15
		0.50	23
		0.25	30
Landing	X to T5	1.00	10
Landing	T5 to T12	0.25	10
		0.50	15
		0.25	20
Landing	T12 to G	0.25	30
		0.50	40
		0.25	50

5.6.5. Tower Controller Object

For this model, the Tower Controller Object (TC) has not been further developed. This object would be defined to represent the behavior of air traffic control behavior in coordinating aircraft traffic for landing, taxi, and takeoff. As defined by the interaction diagram (Figure 5-48), the Tower Controller would provide commands and responses to the Taxi Aircraft Object and runway reservations to the Airport/Airspace Object. In turn, the Tower Controller Object would receive requests and responses from the Taxi Aircraft Object and aircraft positions from the Airport/Airspace Object. Additionally, the Tower Controller Object would receive information on environmental conditions from the Environment Object, including weather, visibility, radio interference, and traffic levels. The attributes for the Tower Controller Object would include communicated information such as runway and taxiway destinations and routes, and hold, proceed and stop commands. The states for the Tower Controller would include a commanding state when the Tower Controller is issuing commands and a monitoring state when the Tower Controller is monitoring aircraft traffic.

Even at this assumed level of detail, the control behaviors exhibited by such a Tower Controller Object represent a relatively coarse abstraction of the actual intricacies of the procedures and behaviors employed to control traffic at a major airport. For example, many airports separate the “tower control” and “ground control” functions such that they are performed by different individuals. A single controller object, as described above, would not be able to model the details of this relationship or discover risks that might arise, for example, in the interactions between these controllers as they hand off aircraft to one another. While a two-object controller model might provide additional insights into aviation safety issues, it would be a significantly more complex model to build and solve.

5.6.6. Environment Object

For this model, the Environment Object (Env) has not been further developed. This object would be defined to set the attributes for the environmental context for the analysis. As defined by the interaction diagram (Figure 5-48) for this example, the Environment Object would provide information about environmental context to the Taxi Aircraft, the Landing Aircraft, and the Tower Controller. The attributes for the Environment Object would include values for visibility, lighting, radio conditions, taxi traffic, and airport signage.

5.6.7. Model Applications

This preliminary model can be applied for several different runway operations including aircraft spacing for runway crossings, and risk assessment for runway collisions, runway incursions and for land and hold short operations. This example problem demonstrates how the OBEST method can be applied for developing system safety models that integrate human performance as well as the operations of and relationships among the numerous components of the complex aviation system environment. The Taxi Aircraft and Landing Aircraft Objects can be extended to incorporate attributes that represent a pilot’s comprehension of assigned instructions as well as flight crew characteristics including performance factors such as experience, site knowledge, mental state, and task load. The environmental context can be developed to define visual cues,

which in combination with the flight crew characteristic can be used to model human performance for recognizing and correcting dangerous situations during runway operations.

5.7. Summary

This chapter has provided six example problems that demonstrate the power and versatility of the OBEST modeling methodology. While many of the problems were relatively simple, they were selected to emphasize both the various features of OBEST as well as the wide variety of problems and analysis domains that can be represented using OBEST. The details of the IRM and DRM models were left out of the discussions in this chapter because of space limitations. However, those models are presented in more detail in the appendices to this report. The curious reader should take the initiative to review those models in detail and reconcile them with the material presented in this chapter.*

After such an involved trip through these six example problems, the reader has undoubtedly thought of other application domains where OBEST might be useful. Clearly, the flexibility that is inherent to the OBEST methodology makes it applicable to numerous classes of problems. One price that is paid for this high degree of flexibility is that it is incumbent upon the user to provide large amounts of information to embody the details of system behavior within the OBEST models. It is our hope, however, that one day we will be able to build libraries of generic OBEST objects to represent frequently encountered components that can be rapidly inserted into OBEST system models. This will help reduce the work required for the analyst to make effective use of this very powerful methodology.

* The material presented in this chapter did not show numerical solutions for these example problems. This is because the OBEST software was not completely functional at the time these models were developed. The models for which the OBEST software was actually applied make use of data that we do not yet have permission to release into the public domain. The results of these OBEST analyses are likely to be published at a later date.

6. Conclusions and Future Directions

6.1. *Conclusions*

The OBEST methodology described in this report provides a new way for looking at scenario based risk assessment problems that has many desirable characteristics while removing a number of the major limitations found in previous methods.

- It provides the capability to explore a universe of probabilistic event scenarios (like an event tree) while removing the limitation of a priori event ordering.
- It provides a capability for variable event ordering (like a discrete event simulation) without requiring computationally inefficient Monte Carlo simulation to infer the probabilistic characteristics of the many scenarios.
- It enables an analyst to describe the behavior of the various components within a system, and let the analytical engine infer the emergent behavior that results from their interactions (like an object-oriented analysis) rather than requiring the analyst to distill the system behavior within the task of risk model construction (as required in an event tree analysis).
- It enables an analyst to model many different types of initial and boundary conditions for a system using a single object model (like a simulation tool) rather than requiring the analyst to construct separate models for each set of system conditions to be considered (as is often required in an event tree analysis).

We have demonstrated that this methodology is applicable to a broad variety of risk assessment problems. In this project we have provided example problems related to race conditions, infrastructure assessment, air traffic safety, fire damage assessment, and even a recreational activity. We noted (see Section 3.6) the potential to apply this methodology for a broad range of infrastructure interdependencies problems (this class of problems provided motivation for the original methodology development activities). We also see immediate applicability for this method in a broad variety of risk assessment domains, including:

- Nuclear reactor safety (OBEST is a potential replacement methodology for accident progression event tree applications).
- Nuclear weapons safety (OBEST could replace current event tree analyses for characterizing the boundary conditions to which a weapon would be subjected during a transportation accident),
- Physical security (OBEST could replace current event tree analyses in certain applications, and may also provide a probabilistic methodology for simulating “force on force” exercises),
- Information assurance (OBEST could provide a probabilistic method to characterize broad classes of information system attack scenarios – and especially the probabilistic aspects of the system’s defensive responses), and

- Aviation safety (the variable event ordering in the OBEST enables probabilistic characterization of many types of aviation safety scenarios, including the already surveyed areas of flight through icing conditions and airport runway transgressions).

The beauty of this methodology is that it is very general and not tightly tailored to any single application domain. We believe that the number of application domains will increase as the methodology gains wider exposure.

In its current form, the OBEST methodology is most applicable to situations where the order of events cannot be determined a priori (thus, making an event tree analysis very challenging). Many of these situations involve race conditions – that is, situations where it is important to know whether event A happens before event B. These situations often come about when a system is making use of a limited and depletable resource such as fuel or water, and an undesirable condition occurs if that resource becomes fully exhausted before the system stops having need of it. The OBEST models these types of race conditions relatively well, and these capabilities will only be improved as the methodology and software are enhanced in the future.

A second class of problems for which OBEST is highly applicable involves assessments where a single system needs to be analyzed under a broad variety of situations. This is especially true if these various situations will cause important events to happen in different orders, because this lack of a consistent ordering will require the development of multiple ETA models – a cumbersome and time-consuming process. Since the OBEST methodology embodies the behavior of the system components, and then coaxes the system behavior from the probabilistic interactions between these components, OBEST enables the analyst to develop a single behavioral model of the system and then subject it to a variety of initial and boundary conditions to represent this broad variety of situations. Thus, the analyst develops the OBEST model once and exercises it many times to characterize these many situations, while they might be required to construct many event tree models and exercise each of them only once to obtain the same result. This characteristic of the OBEST methodology holds the potential to dramatically improve the productivity of the risk analyst.

While OBEST is highly applicable to the classes of problems described above, it is not the most appropriate tool for all applications. OBEST is fundamentally an inductive analysis method that uses a discrete model space in search of a large number of potential system scenarios. Thus, as an inductive method, it is not appropriate for deductive modeling of system failure criteria (i.e., it is not a suitable replacement for FMECA or FTA). As a discrete analysis method, it is unsuitable for modeling systems that cannot be adequately modeled apart from the use of continuous variables (or for systems that would require a large number of discrete intervals to approximate a continuous variable). As a branching methodology, it is inappropriate for modeling systems that exhibit either minimal branching (those would be better analyzed using traditional design of experiments methods) or quasi-continuous branching (the enumeration and categorization of a complete set of scenarios for such problems requires prohibitive amounts of computational effort and storage). Finally, since OBEST requires the analyst to manually encode the behaviors of system components into objects, OBEST may be difficult to apply to systems whose components exhibit a very high degree of behavioral complexity and yet are irreducibly complex (i.e., cannot be broken down into simpler components or subsystems without unacceptably sacrificing model fidelity). However, even with these limitations, the

OBEST methodology provides a powerful new tool for analyzing a broad array of problems that are important to the risk assessment community.

6.2. Recommendations for Future Work

In its current form, the OBEST methodology and software are a little bit like a rough cut diamond. They have a certain amount of obvious value in their current state, but could become dramatically more valuable given some additional refinement. We recommend the following specific steps be taken in future projects to help the OBEST methodology reach its full potential.

Complete the Software. The task of developing the OBEST software proved much more difficult than originally envisioned by the LDRD project team. Therefore, numerous features that were originally envisioned for this software had to be deferred for later implementation. Some of these are cosmetic or convenience features (e.g., many of the Cancel buttons do not function as one would normally expect). However, there are a number of computational features that also remain to be implemented. Bringing the software in line with the original design specification would go a long way toward helping the OBEST methodology gain a wider acceptance.

Graphical Modeling Interface. While the current OBEST software is a Microsoft Windows application, a large amount of its user interface – especially its model development interface – makes use of a text-based paradigm. On the other hand, the examples shown in Chapter 5 make use of a variety of types of *diagrams* to communicate the basic model concepts. The OBEST software should be extended to enable the analyst to develop models using a graphical paradigm to directly create these diagrams.

Uncertainty Analysis Capabilities. The current OBEST software makes use of point estimate values for all of its numeric values. Current Sandia philosophy rightly holds that the analyst should examine the effects of all input parameter uncertainties on the results generated by the model. The SETAC large event tree analysis software incorporates uncertainty analysis capabilities through its interface with the LHS code. SETAC solves the event tree model separately for each observation generated within LHS. A similar capability could easily be implemented in the OBEST software. This should be a high priority item in any future OBEST software development work.

Separate Capacities and Utilization Rates. In its current form, OBEST uses time as a surrogate variable to incorporate both the storage capacity of a consumable commodity and the rate at which that commodity is being consumed. This poses problems when the rate of consumption changes as a function of the changing system conditions. It is conceptually simpler (and more straightforward for the analyst) to explicitly specify the amount of the commodity that is present as one parameter value, and the rate at which that commodity is being consumed as a second parameter value. The computation of the time at which this commodity would be exhausted is extremely simple, although it may be a challenge to develop a rich and appropriate syntax that would allow the analyst the needed flexibility to incorporate these concepts into IRM and DRM case statements. Implementation of this capability would make most OBEST models vastly more scrutable and likely much more realistic. For this reason, this capability should also be a high priority item in any future OBEST software development work.

Branching Based On “Side Computations.” The next step beyond using separate capacities and utilization rates is the ability to make branching decisions in IRM and DRM case statements based on a richer set of mathematical computations. This capability is implemented through the use of “variables” in the SETAC event tree analysis software. A classic example from nuclear power accident progression event tree analysis involves the accumulation of hydrogen gas from multiple sources. In SETAC, one constructs a variable to represent the quantity of hydrogen produced by each source, and then makes branching decisions related to the ignition characteristics for that hydrogen based on the sum of those variable values. The variable values are assigned based on the results of specific questions within the event tree (e.g., if *this* occurs then set the hydrogen produced by *that* process to 10 kg; else set to 100 kg).^{*} A similar capability could be developed for OBEST in which the IRM and DRM scripts would be allowed to set a variable to a particular value and the IRM and DRM case statements would be capable of performing mathematical manipulations on those values to make branching decisions. This would dramatically enhance the power of the OBEST methodology and could make it applicable to whole new classes of problems.

Integrate With Network Analysis Tools. One long-term goal for OBEST development should be to realize the vision of creating hybrid probabilistic-deterministic risk models. The example cited most frequently in this report for such a hybrid model involves linking the OBEST software with an electric power circuit stability analysis tool to perform infrastructure interdependency assessments. Under such a vision, the interactions amongst the various elements within an electric power grid are modeled using a circuit stability analysis tool at each time step in an OBEST analysis. The results from that tool are used to set the conditions within each object in the OBEST model, and OBEST models the effect that these conditions have on other infrastructure entities (e.g. telephone central offices, public safety offices, water and sewer facilities). Should these interactions feedback into changes in entities within the electric power grid, these changes would be reflected in new boundary conditions for the circuit stability tool. This allows both tools to do the parts of the infrastructure analysis problem for which they are best suited and does not force either to be dramatically adapted to model the infrastructure interdependency problem. One could envision a number of model hybridization opportunities that might be implemented with similar success. Clearly, this would require a major research effort that is not likely to be implemented in the near term.

Importance Measures. A key advantage to the fault tree analysis methodology has always been the ability to easily extract information about the importance of particular component failures to

^{*} One key aspect of the power of these variable computations in SETAC is that the actual values to which these variables can be set are part of the uncertainty analysis. Thus, while the process might nominally produce 10 kg of hydrogen, the uncertainty distribution for that parameter might range from 2 kg to 50 kg. By actually performing a mathematical sum, and by allowing similar uncertainty in the thresholds to which this sum is compared, SETAC is able to characterize very complex behavior using very straightforward modeling commands. While this same problem could be done manually by the analyst using discrete attribute values and logical case statements, the number of such case statements expands combinatorially with the number of variables, the number of discrete values, and the number of thresholds. The quality assurance requirements alone for such a modeling method would make it of questionable utility.

the reliability of the overall system. Importance measures such as risk reduction, risk increased, partial derivative, and even cut set order all provide valuable insights to the analyst that are not available from the “plain vanilla” fault tree analysis results. The lack of such importance measures for event tree analysis has been one impediment to that method’s more widespread use. While it is a nontrivial task, research effort should be expended to investigate what types of importance measures one might be able to generate for OBEST models, and how those models might relate back to the widely accepted fault tree importance measures.* Development of such measures would provide important new avenues for analysts to draw insight and understanding regarding the systems that they model.

Interface With Optimization Methods. Another key advantage to fault tree analysis methods is that their results have been easy to interface with optimization methods. One can incorporate a reliability metric (as derived from the cut sets generated by a fault tree analysis) into the utility function for a variety of optimization methods in order to search for optimal system designs, optimal maintenance schedules, optimal spare parts inventories, and so on. It has historically been more difficult to generate metrics for event tree analysis results that are as useful in these types of optimality analyses. In addition, the fact that cut sets can be requantified quickly while event tree analysis results take much longer to regenerate has further restricted the use of event tree analysis in optimization studies. Since OBEST shares many of the solution characteristics of a large event tree model, one would expect similar problems when attempting to use OBEST results in optimization studies. Future research on OBEST should examine how OBEST results can be best incorporated into optimization utility functions as well as the potential for some “shortcut” solution schemes that would make using OBEST a practical alternative in these optimization problems.

These paragraphs lay out a broad array of research opportunities for the OBEST methodology. Clearly there are a number of short-term improvements to the software and methodology that can dramatically improve the usefulness and, hopefully, acceptance of the method. The long-term research directions would enable OBEST to become a truly mature integrating platform with the potential to unite many currently disparate domains in the risk assessment world.

6.3. Summary

The purpose of this report has been to introduce the reader to the OBEST event scenario modeling methodology and to demonstrate its use through several example problems. OBEST uses concepts derived from event tree analysis, object-oriented analysis, and discrete event simulation to enable an analyst to rapidly construct realistic models of scenarios for which an *a priori* discovery of event ordering is either cumbersome or impossible (especially families of scenarios that exhibit inconsistent or variable event ordering). Each scenario that results from an OBEST analysis is automatically associated with an estimate of its likelihood because the

* This research might have the added benefit of providing insights that can be used to develop similar importance measures for the event tree analysis method since they share a common heritage in terms of solution algorithm.

methodology employs probabilistic branching as an integral part of the object model definition. The methodology solves the resultant object models through the use of a recursive algorithm that identifies all possible scenarios and their associated occurrence probabilities. Since the occurrence probabilities are developed directly by the solution algorithm, they need not be computed by statistical inference from a large sample of Monte Carlo observations (as is common in some discrete event simulation methodologies). Thus, OBEST is not only much more computationally efficient than these discrete event simulation methods, but it also discovers scenarios that have extremely low probabilities as a natural analytical result – scenarios that would likely be missed by a Monte Carlo-based method. While the ultimate completion and enhancement of the OBEST software tool for production use would dramatically increase the utility of the OBEST technique, the methodology and demonstration software developed under this project provide a basis for significant improvements in the way risk assessment studies are performed both now and in the future.

7. References

1. Wyss, Gregory D., "Risk Assessment and Risk Management for Energy Applications," in *Energy 2000: State of the Art*, ed. Peter Catania, Balaban Publishers, L'Aquila, Italy, pp. 163-184 (2000).
2. Roberts, N.H., Vesely, W.E., Haasl, D.F., and Goldberg, F.F., *Fault Tree Handbook*, NUREG-0492, U.S. Nuclear Regulatory Commission, Washington, DC, 1981.
3. Beaver, C., Schroepfel, R., and Snyder, L., "A Design for Anonymous Authenticated Information Sharing," To appear in *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance Workshop*, IEEE Press, New York, 2001.
4. Painton, L., and Campbell, J., "Genetic Algorithms in the Optimization of System Reliability," *IEEE Transactions on Reliability*, Special Issue on Design, **44**(2):172-178, 1995.
5. Basu, N., Pryor, R.J., and Quint, T. *Aspen - A Microsimulation Model for the Economy*, SAND96-1225A, Sandia National Laboratories, Albuquerque, NM, 1996.
6. Barton, D.C and Stamber, K.L. "An Agent-Based Microsimulation of Critical Infrastructure Systems," in *Energy 2000: The Beginning of a New Millennium*, ed. Peter Catania, Balaban Publishers, L'Aquila, Italy, pp. 709 - 714 (2000).
7. Gauntt, R.O., *et al.*, *MELCOR Computer Code Manuals – Version 1.8.4*, NUREG/CR-6119, SAND97-2398, Vols.1-2, Rev.1, prepared by Sandia National Laboratories, Albuquerque, NM, for the U.S. Nuclear Regulatory Commission, Washington, DC, July 1997 (Revised May 1998).
8. Polif, T., and Sathyanarayana, A., "Efficient Algorithms for Reliability Analysis of Planar Networks – A Survey," *IEEE Trans. Reliability*, Vol. R-35, No. 3, pp. 252-259, August 1986.
9. Wood, R.K., "Factoring Algorithms for Computing K-Terminal Network Reliability," *IEEE Trans. Reliability*, Vol. R-35, No. 3, pp. 269-278, August 1986.
10. Karger, D.R., "Random Sampling in Cut, Flow and Network Design Problems," in *Proceedings of the 26th ACM Symposium on Theory of Computing*, pp. 648-657, ACM Press, May 1994.
11. Karger, D.R., "A Randomized Fully Polynomial Approximation Scheme for the All-Terminal Network Reliability Problem," in *Proceedings of the 27th ACM Symposium on Theory of Computing*, pp. 11-17, ACM Press, May 1995.

12. Wyss, Gregory D., *A Mathematical Foundation for the Development of Cut Sets for Arbitrarily Interconnected Networks*, SAND97-1967 (Limited distribution – patent caution), Sandia National Laboratories, Albuquerque, NM, August 1997.
13. Kleinrock, L., *Queuing Systems Volume 1: Theory*, John Wiley & Sons, New York, 1975.
14. Frost, V.F., and Melamed, B., “Traffic Modeling for Telecommunications Networks,” *IEEE Communications Magazine*, pp. 70-81, March 1994.
15. Singh, SN, and Srivastava, SC, “Improved contingency selection algorithm for voltage security analysis,” *Electric Machines and Power Systems*, **26**:8(855-871), October 1998.
16. McCormick, N.J., *Reliability and Risk Analysis: Methods and Nuclear Power Applications*, Academic Press, New York, 1981.
17. Greenberg, H.R., and Cramer, J.J., editors, *Risk Assessment and Risk Management for the Chemical Process Industry*, Van Nostrand Reinhold, New York, 1991.
18. Cramond, W.R., *et al.*, *Probabilistic Risk Assessment Course Documentation*, SAND85-1495, NUREG/CR-4350, 7 volumes, prepared by Sandia National Laboratories, Albuquerque, NM, for the U.S. Nuclear Regulatory Commission, Washington, DC, 1985.
19. Hays, K.M., *A User’s Guide to GAETR: Sandia’s “Graphical Analysis of Event Trees” Software*, SAND97-2097, Sandia National Laboratories, Albuquerque, NM, September 1997.
20. Griesmeyer, J.M., and Smith, L.N., *A Reference Manual for the Event Progression Analysis Code (EVNTRE)*, SAND88-1607, NUREG/CR-5174, Prepared by Sandia National Laboratories, Albuquerque, NM, for the U.S. Nuclear Regulatory Commission, Washington, D.C., September 1989.
21. Jae, M., and Apostolakis, G.E., “The Use of Influence Diagrams for Evaluating Severe Accident Management Strategies,” *Nuclear Technology* **99**:142-157, August 1992.
22. Jansma, R.M., *et al.*, *Risk-Based Assessment of the Surety of Information Systems*, SAND96-2027, Sandia National Laboratories, Albuquerque, NM, July 1996.
23. Wyss, G.D., *The Formulation and Solution of Time-Variant Markov Models for Probabilistic Risk Analysis*, Master’s Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1985.
24. Devooght, J., and Smidts, C., “Probabilistic Reactor Dynamics – I: The Theory of Continuous Event Trees,” *Nuclear Science and Engineering* **111**:229-240, 1992.

25. Smitds, C., and Devooght, J., "Probabilistic Dynamics: A Numerical Comparison Between a Continuous Event Tree and DYLAM-Type Event Tree," from the Proceedings of PSAM II, G.E. Apostolakis and J.S. Wu of the University of California at Los Angeles, editors, San Diego, California, March 20-25, 1994.
26. Devooght, J., and Smitds, C., "Probabilistic Reactor Dynamics – III: A Framework for Time-Dependent Interaction Between Operator and Reactor During a Transient Involving Human Error," *Nuclear Science and Engineering* **112**:100-113, 1992.
27. Iman, R.L., Helton, J.C., and Johnson, J.D., "A Methodology for Grouping Source Terms for Consequence Calculations in Probabilistic Risk Assessments," *Risk Analysis* **10**(4):507-520, 1990.
28. Wyss, G.D., and Jorgensen, K.H., *A User's Guide to LHS: Sandia's Latin Hypercube Sampling Software*, SAND98-0210, Sandia National Laboratories, Albuquerque, NM, February 1998.
29. Wyss, G.D., *Cassini Spacecraft Uncertainty Analysis Data and Methodology Review and Update: Volume 2 – A Technical Description of the Sampling Methods Employed in the Cassini Uncertainty Analysis*, SAND2000-1764, Volume 2, Sandia National Laboratories, Albuquerque, NM, July 2000.
30. Press, W.H., et al., *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd ed., Cambridge University Press, Cambridge, 1992.
31. Cochran, W.G., *Sampling Techniques*, 2nd ed., Wiley, New York, 1963.
32. Robinson, D.G., *A Survey of Probabilistic Methods Used in Reliability, Risk and Uncertainty Analysis: Analytical Techniques I*, SAND98-1189, Sandia National Laboratories, Albuquerque, NM, June 1998.
33. Siu, N., "Risk Assessment for Dynamic Systems: An Overview," *Reliability Engineering and System Safety*, **43**(1):43-73, 1994.
34. Acosta, C., and Siu, N., "Dynamic Event Trees in Accident Sequence Analysis: Application to Steam Generator Tube Rupture," *Reliability Engineering and System Safety*, **41**(2):135-154, 1993.
35. NUS Corporation, *Final Safety Analysis Report for the Ulysses Mission*, ULS-FSAR-006, Prepared for the U.S. Department of Energy Office of Special Applications, Washington, DC, 1990.
36. U.S. Federal Aviation Administration Technical Center, *Airport and Airspace Delay Simulation Model (SIMMOD)*, <http://www.tc.faa.gov/act-500/nasacb/simmod.html>, Atlantic City, New Jersey, 2000.

37. Abkin, M.H., personal communications regarding ATAC Corporation's use of the SIMMOD Pro! Simulation software, ATAC Corporation, 757 N. Mary Avenue, Sunnyvale, California, October 2000.
38. Bobick, J.C., Abkin, M.H., *et al.*, *Development of Fast-Time Simulation Techniques to Model Safety Issues in the National Airspace System: Phase I Final Report*, Prepared for the National Aeronautics and Space Administration Ames Research Center by ATAC Corporation, Sunnyvale, California, December 1999.
39. Wyss, G.D., Craft, R.L., and Funkhouser, D.L., *The Use of Object-Oriented Analysis Methods in Surety Analysis*, SAND99-1242, Sandia National Laboratories, Albuquerque, NM, May 1999.
40. Shlaer, S., and Mellor, S.J., *Object Lifecycles: Modeling the World in States*, Prentice Hall, Englewood Cliffs, N. J., 1992.
41. Booch, G., Jacobson, I., and Rumbaugh, J., *The Unified Modeling Language User Guide*, Addison-Wesley, New York, 1998.
42. Ziegler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic Press, San Diego, California, 1990.
43. Kuipers, B., *Qualitative Reasoning*, The MIT Press, Cambridge, MA, 1994.
44. Griesmeyer, J.M., and Smith, L.N., *A Reference Manual for the Event Progression Analysis Code (EVNTRE)*, SAND88-1607, NUREG/CR-5174, Prepared for the U.S. Nuclear Regulatory Commission by Sandia National Laboratories, Albuquerque, New Mexico, September 1989.
45. Wyss, Gregory D., "Training Materials for the SETAC Event Tree Analysis Software," Memo to Susan Dingman, Sandia National Laboratories, Albuquerque, New Mexico, September 5, 2000.
46. Firebaugh, M.W., *Artificial Intelligence: A Knowledge-Based Approach*, Boyd and Fraser Publishing Co., Boston, 1988.
47. U.S. Nuclear Regulatory Commission, *Severe Accident Risks: An Assessment for Five U.S. Nuclear Power Plants*, NUREG-1150, U.S. Nuclear Regulatory Commission, Washington, D.C., 1990.
48. Ericson, D.M., *et al.*, *Analysis of Core Damage Frequency: Internal Events Methodology*, NUREG/CR-4550, SAND86-2084, Prepared for the U.S. Nuclear Regulatory Commission by Sandia National Laboratories, Albuquerque, New Mexico, 1989.

49. Gorham, E.D., *et al.*, *Evaluation of Severe Accident Risks: Methodology for the Containment, Source Term, Consequence, and Risk Integration Analyses*, NUREG/CR-4551, SAND86-1309, Prepared for the U.S. Nuclear Regulatory Commission by Sandia National Laboratories, Albuquerque, New Mexico, 1993.
50. U.S. Nuclear Regulatory Commission, *A Review of NRC Staff Uses of Probabilistic Risk Assessment*, NUREG-1489, U.S. Nuclear Regulatory Commission, Washington, D.C., 1994.
51. Payne, A.C., Jr., and Camp, A.L., *Parametric HECTR Calculations of Hydrogen Transport and Combustion at N Reactor*, SAND86-2630, Sandia National Laboratories, Albuquerque, New Mexico, June 1987.
52. Wyss, G.D., *et al.*, *Accident Progression Event Tree Analysis for Postulated Severe Accidents at N Reactor*, SAND89-2100, Sandia National Laboratories, Albuquerque, New Mexico, June 1990.
53. Miller, L.A., *et al.*, *N Reactor Probabilistic Risk Assessment Supporting Calculations*, SAND89-2101, 3 Volumes, Sandia National Laboratories, Albuquerque, New Mexico, June 1990.
54. Camp, A.L., *et al.*, *Level III Probabilistic Risk Assessment for N Reactor*, WHC-MR-0045, SAND89-2102, Prepared for Westinghouse Hanford Corporation by Sandia National Laboratories, Albuquerque, New Mexico, April 1990.

This Page Intentionally Blank

A. Example Details for Electric Power Supply for a Police Station

This appendix provides a description of the system structure, interactions, and state transition diagrams for the electric power supply for a police station presented in Section 5.1 of the main body of the report. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams.

A.1 System Structure and Interactions

The system structure diagram for this example problem is shown in Figure A-1. The description of the functional blocks is provided in Table A-1. The interaction diagram is provided in Figure A-2. The functional interactions between the blocks are indicated by directed arrows and are specified in Table A-2.

Figure A-1. System structure diagram for police station power model.

Table A-1. Functional blocks for police station power.

Telephone Switch Building (TSB): Provides telephone service for the police station and for electric power company.
Police Station Power (PSP): Normal and backup power for the police station to support community police service.
Electric Power Bus (EPB): Provides power for the telephone switch building and police station.

Figure A-2. Interaction diagram for police station power model.

Table A-2. Functional interactions for police station power model.

TSB-EPB:	The telephone switch building provides telephone service to support the electric power bus.
TSB-PS:	The telephone switch building provides telephone service to the police station.
EPB-TSB:	The electric power bus provides normal power to the telephone switch building.
EPB-PS:	The electric power bus provides normal power to the police station.

A.2 Electric Power Bus Object

The Electric Power Bus Object (EPB) provides normal electric power for the telephone switch building and the police station. The state transition diagram for the electric power bus is shown in Figure A-3. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Power fails* can be generated internally or externally to the object and causes a transition from State:(Provides Power) to State:(Does Not Provide Power).

Event:*Power restored* can be generated internally or externally to the object and causes a transition from State:(Does Not Provide Power) to State:(Provides Power).

Attributes (Values)

Power Provided – PP (Provides Power, Does Not Provide Power)

Figure A-3. State transition diagram for the electric power bus.

Initial Conditions

Initial state is (Provides Power) until analyst generates Event:*Power fails*. The initial value of attribute PP=(Provides Power).

State:(Provides Power)

Immediate Response Model:

If PP=(Provides Power) Then Set PP=(Provides Power)

Delayed Response Model:

None.

State:(Does Not Provide Power)

Immediate Response Model:

If PP=(Does Not Provide Power) Then Set PP=(Does Not Provide Power)

Delayed Response Model:

If TSB:TA=(Provides Telephone Service) Then

P=0.90, Delay 2 Hours, then Generate Event:*Power restored*

P=0.10, Delay 36 Hours, then Generate Event:*Power restored*

Else

P=1.00, Delay 36 Hours, then Generate Event:*Power restored*

A.3 Telephone Switch Building Object

The Telephone Switch Building Object (TSB) provides telephone service for the police station and to support restoration of the electric power bus. The state transition diagram for the telephone switch building is shown in Figure A-4. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Power fails* can be generated internally or externally to the object and causes a transition from State:(Available With Normal Power) to State:(Available With Generator Power).

Event:*Generator fails* is generated internally to the object and causes a transition from State:(Available With Generator Power) to State:(Available With Battery Power).

Event:*Batteries fail* is generated internally to the object and causes a transition from State:(Available with Battery Power) to State:(Unavailable).

Figure A-4. State transition diagram for telephone switch building.

Event:*Power restored* can be generated internally or externally to the object and causes the following transitions:

from State:(Available With Generator Power) to State:(Available With Normal Power);
 from State:(Available With Battery Power) to State:(Available With Normal Power); and
 from State:(Unavailable) to State:(Available With Normal Power).

Attributes (Values)

Telephone Available – TA (Provides Telephone Service, Does Not Provide Telephone Service)
 Switch Mode – SM (Available With Normal Power, Available With Generator Power, Available With Battery Power, Unavailable).

Initial Conditions

Initial state is (Available With Normal Power) until analyst generates Event:*Power fails*. The initial values of object attributes are TA=(Provides Telephone Service) and SM=(Available With Normal Power).

State:(Available With Normal Power)

This is the initial state.

Immediate Response Model:

If TA/=(Provides Telephone Service) Then Set TA=(Provides Telephone Service)
 Else If SM/=(Available With Normal Power) Then Set SM=(Available With Normal Power)

Delayed Response Model:

None.

State:(Available With Generator Power)

Immediate Response Model:

If SM/=(Available With Generator Power) Then

This indicates that the object has just entered this state – it is only entered once.

P=0.03, Generate Event:*Generator fails*.

P=0.97, Set SM=(Available With Generator Power)

Else If TA/=(Provides Telephone Service) Then

Set TA=(Provides Telephone Service)

Delayed Response Model:

Always Do This

Delay 48 hours, Generate Event:*Generator fails*.

State:(Available With Battery Power)

Immediate Response Model:

If TA/=(Provides Telephone Service) Then Set TA=(Provides Telephone Service)

Else If SM/=(Available With Battery Power) Then Set SM=(Available With Battery Power)

Delayed Response Model:

Always Do This

Delay 8 hours, then generate Event:*Battery fails*.

State:(Unavailable)

Immediate Response Model:

If TA/=(Does Not Provide Telephone Service) Then Set TA=(Does Not Provide Telephone Service)

Else If SM/=(Unavailable) Then Set SM=(Unavailable)

Delayed Response Model:

None.

A.4 Police Station Power Object

The Police Station Power Object (PSP) represents normal and backup power for the police station to support community police service. The state transition diagram for the telephone switch building is shown in Figure A-5. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure A-5. State transition diagram for police station.

Events and State Transitions

Event:*Power fails* can be generated internally or externally to the object and causes a transition from State:(Available with Normal Power) to State:(Available with Generator Power).

Event:*Police generator fails* is generated internally to the object and causes a transition from State:(Available with Generator Power) to State:(Unavailable).

Event:*Power restored* can be generated internally or externally to the object and causes the following transitions:

from State:(Available with Generator Power) to State:(Available with Normal Power);

and

from State:(Unavailable) to State:(Available with Normal Power).

Attributes (Values)

Station Availability – SA (Available With Normal Power, Available With Generator Power, Unavailable).

Initial Conditions

Initial state is (Available with Normal Power) until analyst generates Event:*Power fails*. The initial value of attribute SA=(Available With Normal Power).

State:(Available with Normal Power)

This is the initial state.

Immediate Response Model:

If SA/=(Available With Normal Power) then Set SA=(Available With Normal Power)

Delayed Response Model:

None.

State:(Available with Generator Power)

Immediate Response Model:

If SA/=(Available With Generator Power) then

This indicates that the object has just entered this state – it is only entered once.

P=0.05, Generate Event:*Police Generator fails.*

P=0.95, Set SA=(Available With Generator Power)

Delayed Response Model:

If TSB:TA=(Provides Telephone Service) then

P=0.90, Delay 48 hours (*at least one day's additional generator fuel is obtained*)

P=0.10, Delay 24 hours then generate Event:*Generator fails.*

Else

Delay 24 hours, then generate Event:*Generator fails.*

State:(Unavailable)

Immediate Response Model:

If SA/=(Unavailable) then Set SA=(Unavailable)

Delayed Response Model:

None

This Page Intentionally Blank

B. Example Details for Fuel Tank Supply for a Gas Burner

This appendix provides a description of the system structure, interactions, and state transition diagrams for the fuel tank supply for a gas burner presented in Section 5.2 of the main body of the report. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams.

B.1 System Structure and Interactions

An illustration of the fuel tank supply system is shown in Figure B-1. The system structure diagram for this example problem is shown in Figure B-2. The description of the functional blocks is provided in Table B-1. The interaction diagram is provided in Figure B-3. The functional interactions between the blocks are indicated by directed arrows and are specified in Table B-2.

Figure B-1. Fuel tank supply for a gas burner.

Figure B-2. System structure diagram for gas burner fuel supply model.

Table B-1. Functional blocks for gas burner fuel supply.

Valve: Opens and closes to control fuel flow for the gas burner.
Fuel System: Five tanks that comprise fuel system for gas burner.

Figure B-3. Interaction diagram for gas burner fuel supply model.

Table B-2. Functional interactions for gas burner fuel supply model.

Valve - Fuel System: The valve requests a specified fuel flow from the fuel system.
Fuel System - Valve: The fuel system provides fuel supply in response to the valve's requests.

B.2 Valve Object

The Valve Object represents a valve that opens and closes to control fuel flow for the gas burner. The state transition diagram for the valve is shown in Figure B-4. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Turn On* is generated by the Valve Object, and also externally by the analyst to begin the analysis, and causes a transition from State:(Off) to State:(On).

Event:*Turn Off* is generated by the Valve Object and causes a transition from State:(Off) to State:(On).

Figure B-4. State transition diagram for the valve.

Attributes (Values)

FlowRequest – FR (0 gpm, 2 gpm)

Initial Conditions

State:(Off) (use “TurnOn” to start the analysis); FR=(0)

State:(Off)

Immediate Response Model:

Set FR=(0), Turn on my clock

Delayed Response Model:

If Fuel System:FL=(0) then

Do Nothing

Else

P=0.3, Delay 10, Reset my timer, Generate Event:*TurnOn*

P=0.4, Delay 20, Reset my timer, Generate Event:*TurnOn*

P=0.3, Delay 30, Reset my timer, Generate Event:*TurnOn*

State:(Off)

Immediate Response Model:

Set FR=(2), Turn on my clock

Delayed Response Model:

Always do this:

P=0.2, Delay 10, Reset my timer, Generate Event:*TurnOn*

P=0.3, Delay 20, Reset my timer, Generate Event:*TurnOn*

P=0.4, Delay 30, Reset my timer, Generate Event:*TurnOn*

P=0.1, Delay 40, Reset my timer Generate Event:*TurnOn*

B.3 Fuel System Object

The Fuel System Object represents a fuel system consisting of five tanks (a tank farm), including one 10-gallon tank, one 15-gallon tank, and three 25-gallon tanks that provides fuel for the gas burner. The state transition diagram for the fuel system is shown in Figure B-5. The events and

state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure B-5. State transition diagram for fuel system.

Events and State Transitions

Event:*Full* is generated by the Fuel System Object and causes a transition from State:(Refilling) to State:(Gas).

Event:*Change* is generated by the Fuel System Object and causes a transition from State:(Gas) to State:(Tank Change).

Event:*New Tank* is generated by the Fuel System Object and causes a transition from State:(Tank Change) to State:(Gas).

Event:*Drained* is generated by the Fuel System Object and causes a transition from State:(Tank Change) to State:(Empty).

Event:*Refill* is generated by the Fuel System Object and causes a transition from State:(Empty) to State:(Refilling).

Attributes (Values)

Fuel Level – FL (100 gal, 75 gal, 50 gal, 25 gal, 10 gal, 0 gal)
 {Corresponds to 5, 4, 3, 2, 1, or 0 tanks contain any fuel.}

Initial Conditions

State:(Gas); FL=(100)

State:(Refilling)Immediate Response Model:

Set FL=(100), Reset my clock, and Generate Event:*Full*

Delayed Response Model:

None.

State:(Empty)Immediate Response Model:

Set FL=(0), Reset my clock

Delayed Response Model:

None.

State:(Gas)Immediate Response Model:

If Valve:FR=(0) then

 Stop my clock

Else Turn on my clock

Delayed Response Model:

If FL=(100) then

 Delay 12.5, Generate Event:*Change*

Else If FL=(75) then

 Delay 12.5, Generate Event:*Change*

Else If FL=(50) then

 Delay 12.5, Generate Event:*Change*

Else If FL=(25) then

 Delay 7.5, Generate Event:*Change*

Else If FL=(10) then

 Delay 5.0, Generate Event:*Change*

Else (*FL=(0)*)

 No Operation

State:(Tank Change)

Immediate Response Model:

If FL=(100) then
 Set FL=(75), Reset my clock, Generate Event:*NewTank*
Else If FL=(75) then
 Set FL=(50), Reset my clock, Generate Event:*NewTank*
Else If FL=50 then
 Set FL=(25), Reset my clock, Generate Event:*NewTank*
Else If FL=25 then
 Set FL=(10), Reset my clock, Generate Event:*NewTank*
Else If FL=(10) then
 Set FL=(0), Reset my clock, Generate Event:*Drained*
Else (*FL=(0)*)
 Reset my clock, Drained

Delayed Response Model:

None.

C. Example Details for A Singles Racquetball Game

This appendix provides a description of the system structure, interactions, and state transition diagrams for the singles racquetball game presented in Section 5.3 of the main body of the report. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams.

C.1 System Structure and Interactions

The system structure diagram for this example problem is shown in Figure C-1. The description of the functional blocks is provided in Table C-1. The interaction diagram is provided in Figure C-2. The functional interactions between the blocks are indicated by directed arrows and are specified in Table C-2.

Figure C-1. System structure diagram for model of a singles racquetball match.

Table C-1. Functional blocks for singles racquetball game.

Player A (A): One player in a racquetball singles game with an individual serve and shot profile.
Player B (B): A second player in a racquetball singles game with an individual serve and shot profile.
Game (G): A singles game of racquetball.

Figure C-2. Interaction diagram for model of a singles racquetball match.

Table C-2. Functional interactions for model of singles racquetball match.

A-B:	Player A hits shot returns and serves to Player B.
A-G:	Player A provides its status to the Game.
B-A:	Player B hits shot returns and serves to Player A.
B-G:	Player B provides its status to the Game.
G-A:	The Game tracks serving and scoring for Player A.
G-B:	The Game tracks serving and scoring for Player B.

C.2 Player A Object

The Player A Object (A) represents one player in a singles racquetball game. Attributes for this object are defined to provide information about Player A's shot, serve and serve return profiles. The state transition diagram for Player A is shown in Figure C-3. The events and state transitions, attributes, initial conditions, and sets of "if – then – else if – else" scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Start new rally* is generated by the Game Object (G) and causes transition from State:(Stop) to State:(Waiting to/for Serve).

Event:*Player A faults 1st serve* is generated within (A) and causes transition from State:(Waiting to/for Serve) to State:(Waiting to Serve2).

Event:*Player A faults 2nd serve* is generated within (A) and causes transition from State:(Waiting to Serve2) to State:(Stop).

Figure C-3. State transition diagram for Player A object.

Event:*Player A makes shot* is generated within (A) and causes the following transitions:

from State:(Waiting to/for Serve) to State:(Waiting to Return);
 from State:(Waiting to Serve2) to State:(Waiting to Return); and
 from State:(Making a Shot) to State:(Waiting to Return).

Event:*Player A misses shot* is generated within (A) and cause transition from State:(Making a Shot) to State:(Stop).

Event:*Player B makes shot* is generated by (B) and causes the following transitions:

from State:(Waiting to/for Serve) to State:(Making a Shot); and
 from State:(Waiting to Return) to State:(Making a Shot).

Event:*Player B faults 2nd serve* is generated by (B) and causes transition from State:(Waiting to/for Serve) to State:(Stop).

Event:*Player B misses shot* is generated by (B) and causes transition from State:(Waiting to Return) to State:(Stop).

Attributes (Values)

SHOT_TYPE (Drive, Lob, ZDrive, ZLob, Fault, Out, Return, Unset)

SERVE_RETURN (Down the Line, Ceiling, Crosscourt/Pinch, Skip, Unset)

Initial Conditions

SHOT_TYPE=(Unset); SERVE_RETURN=(Unset)

State:(Stop)

The event to transition from this state is generated external to this object (Event:*Start new rally* is generated by (G)).

Immediate Response Model:

None.

Delayed Response Model:

None.

State:(Waiting to/for Serve)Immediate Response Model:

If SERVE=(A), then

P=0.72, SHOT_TYPE=(Drive), Generate Event:*Player A makes shot.*

P=0.03, SHOT_TYPE=(Lob), Generate Event:*Player A makes shot.*

P=0.03, SHOT_TYPE=(ZDrive), Generate Event:*Player A makes shot.*

P=0.01, SHOT_TYPE=(ZLob), Generate Event:*Player A makes shot.*

P=0.21, SHOT_TYPE=(Fault), Generate Event:*Player A faults 1st serve.*

Delayed Response Model:

None.

State:(Waiting to Serve2)Immediate Response Model:

If TRUE, then

P=0.05, SHOT_TYPE=(Drive), Generate Event:*Player A makes shot.*

P=0.84, SHOT_TYPE=(Lob), Generate Event:*Player A makes shot.*

P=0.03, SHOT_TYPE=(ZLob), Generate Event:*Player A makes shot.*

P=0.08, SHOT_TYPE=(Out), Generate Event:*Player A faults 2nd serve.*

Delayed Response Model:

None.

State:(Making a Shot)Immediate Response Model:

If SERVE=(A) then

P=0.80, Set A:SHOT_TYPE=(Return), Generate Event:*Player A makes shot.*

P=0.20, Set A:SHOT_TYPE=(Out), Generate Event:*Player A misses shot.*

Else If B:SHOT_TYPE=(Drive) then

P=0.45, A:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player A makes shot.*

P=0.35, A:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player A makes shot.*

P=0.05, A:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player A makes shot.*

P=0.15, A:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player A misses shot.*

Else If B:SHOT_TYPE=(Lob) then

P=0.25, A:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line) Generate Event:*Player A makes shot.*

P=0.60, A:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player A makes shot.*

P=0.15, A:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player A makes shot.*

P=0.05, A:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player A misses shot.*

Else If B:SHOT_TYPE=(ZDrive) then

P=0.40, A:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player A makes shot.*

P=0.30, A:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player A makes shot.*

P=0.05, A:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player A makes shot.*

P=0.25, A:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player A misses shot.*

Else If B:SHOT_TYPE=(ZLob) then

P=0.20, A:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player A makes shot.*

P=0.50, A:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player A makes shot.*

P=0.25, A:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player A makes shot.*

P=0.05, A:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player A misses shot.*

Else If B:SHOT_TYPE=(Return) then

P=0.80, Set A:SHOT_TYPE=(Return), Generate Event:*Player B makes shot.*

P=0.20, Set A:SHOT_TYPE=(Out), Generate Event:*Player B misses shot.*

Delayed Response Model:

None.

State:(Waiting to Return)

The events to transition from this state is generated external to this object (Event:*Player B makes shot* and Event:*Player B misses shot* are generated by (B)).

Immediate Response Model:

None.

Delayed Response Model:

None.

C.3 Player B Object

The Player B Object (B) represents one player in a singles racquetball game. Attributes for this object are defined to provide information about Player B's shot, serve and serve return profiles. The state transition diagram for Player B is shown in Figure C-4. The events and state transitions, attributes, initial conditions, and sets of "if – then – else if – else" scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Start new rally* is generated by the Game Object (G) and causes transition from State:(Stop) to State:(Waiting to/for Serve).

Event:*Player B faults 1st serve* is generated within (B) and causes transition from State:(Waiting to/for Serve) to State:(Waiting to Serve2).

Event:*Player B faults 2nd serve* is generated within (B) and causes transition from State:(Waiting to Serve2) to State:(Stop).

Event:*Player B makes shot* is generated within (B) and causes the following transitions:
from State:(Waiting to/for Serve) to State:(Waiting to Return);
from State:(Waiting to Serve2) to State:(Waiting to Return); and
from State:(Making a Shot) to State:(Waiting to Return).

Event:*Player B misses shot* is generated within (B) and cause transition from State:(Making a Shot) to State:(Stop).

Event:*Player A makes shot* is generated by (A) and causes the following transitions:
from State:(Waiting to/for Serve) to State:(Making a Shot); and
from State:(Waiting to Return) to State:(Making a Shot).

Event:*Player A faults 2nd serve* is generated by (A) and causes transition from State:(Waiting to/for Serve) to State:(Stop).

Event:*Player A misses shot* is generated by (A) and causes transition from State:(Waiting to Return) to State:(Stop).

Figure C-4. State transition diagram for Player B object.

Attributes (Values)

SHOT_TYPE (Drive, Lob, ZDrive, ZLob, Fault, Out, Return, Unset)

SERVE_RETURN (Down the Line, Ceiling, Crosscourt/Pinch, Skip, Unset)

Initial Conditions

SHOT_TYPE=(Unset); SERVE_RETURN=(Unset)

State:(Stop)

The event to transition from this state is generated external to this object (Event:Start new rally is generated by (G)).

Immediate Response Model:

None.

Delayed Response Model:

None.

State:(Waiting to/for Serve)Immediate Response Model:

If SERVE=(B), then

P=0.63, SHOT_TYPE=(Drive), Generate Event:*Player B makes shot.*

P=0.05, SHOT_TYPE=(Lob), Generate Event:*Player B makes shot.*

P=0.04, SHOT_TYPE=(ZDrive), Generate Event:*Player B makes shot.*

P=0.02, SHOT_TYPE=(ZLob), Generate Event:*Player B makes shot.*

P=0.25, SHOT_TYPE=(Fault), Generate Event:*Player B faults 1st serve.*

Delayed Response Model:

None.

State:(Waiting to Serve2)Immediate Response Model:

If TRUE, then

P=0.03, SHOT_TYPE=(Drive), Generate Event:*Player B makes shot.*

P=0.80, SHOT_TYPE=(Lob), Generate Event:*Player B makes shot.*

P=0.02, SHOT_TYPE=(ZLob), Generate event Event:*Player B makes shot.*

P=0.05, SHOT_TYPE=(Out), Generate Event:*Player B faults 2nd serve.*

Delayed Response Model:

None.

State:(Making a Shot)Immediate Response Model:

If SERVE=(B) then

P=0.75, Set B:SHOT_TYPE=(Return), Generate Event:*Player B makes shot.*

P=0.25, Set B:SHOT_TYPE=(Out), Generate Event:*Player B misses shot.*

Else If A:SHOT_TYPE=(Drive) then

P=0.48, B:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player B makes shot.*

P=0.26, B:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player B makes shot.*

P=0.08, B:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player B makes shot.*

P=0.18, B:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player B misses shot.*

Else If A:SHOT_TYPE=(Lob) then
P=0.18, B:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line) Generate Event:*Player B makes shot.*
P=0.64, B:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player B makes shot.*
P=0.16, B:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player B makes shot.*
P=0.02, B:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player B misses shot.*

Else If A:SHOT_TYPE=(ZDrive) then
P=0.45, B:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player B makes shot.*
P=0.15, B:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player B makes shot.*
P=0.02, B:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player B makes shot.*
P=0.33, B:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player B misses shot.*

Else If A:SHOT_TYPE=(ZLob) then
P=0.30, B:SHOT_TYPE=(Return), SERVE_RETURN=(Down the Line), Generate Event:*Player B makes shot.*
P=0.40, B:SHOT_TYPE=(Return), SERVE_RETURN=(Ceiling), Generate Event:*Player B makes shot.*
P=0.20, B:SHOT_TYPE=(Return), SERVE_RETURN=(Crosscourt/Pinch), Generate Event:*Player B makes shot.*
P=0.10, B:SHOT_TYPE=(Out), SERVE_RETURN=(Skip), Generate Event:*Player B misses shot.*

Else If A:SHOT_TYPE=(Return) then
P=0.75, Set B:SHOT_TYPE=(Return), Generate Event:*Player B makes shot.*
P=0.25, Set B:SHOT_TYPE=(Out), Generate Event:*Player B misses shot.*

Delayed Response Model:

None.

State:(Waiting to Return)

The events to transition from this state is generated external to this object (Event:*Player A makes shot* and Event:*Player A misses shot* are generated by (A)).

Immediate Response Model:

None.

Delayed Response Model:

None.

C.4 The Game Object

The Game Object (G) represents the game played between two racquetball players. Attributes for this object are defined to assign the players' serve and to track the score during the game. The state transition diagram for the Game is shown in Figure C-5. The events and state transitions, attributes, initial conditions, and sets of "if – then – else if – else" scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*Start game* is generated external to (G) to initiate the game and causes the transition from State:(Idle) to State:(Starting Game).

Event:*Start new rally* is generated within (G) and causes the following transitions:
from State:(Starting Game) to State:(Waiting);
from State:(Player A Wins Rally) to State:(Waiting); and
from State:(Player B Wins Rally) to State:(Waiting).

Event:*Player A misses shot* is generated from (A) and causes the transition from State:(Waiting) to State:(Player B Wins Rally).

Event:*Player B misses shot* is generated from (B) and causes the transition from State:(Waiting) to State:(Player A Wins Rally).

Event:*Player A faults 2nd serve* is generated by (A) and causes the transition from State:(Waiting) to State:(Player B Wins Rally).

Event:*Player B faults 2nd serve* is generated by (B) and causes the transition from State:(Waiting) to State:(Player A Wins Rally).

Event:*End game* is generated within (G) and causes the following transitions:
from State:(Player A Wins Rally) to State:(Idle).
from State:(Player B Wins Rally) to State:(Idle).

Attributes (Values)

SERVE (A, B)

SHOT (A, B, 0)

SCORE_A (Integer = 0-15)

SCORE_B (Integer = 0-15)

WINNER (A, B, Unknown)

Initial Conditions

Initial conditions are set within State:(Starting Game) after Event:*Start game* is generated.

SERVE=(A); SHOT=(0); SCORE_A=(0); SCORE_B=(0); WINNER=(Unknown)

Figure C-5. State transition diagram for Game Object.

State:(Idle)

The event to transition from this state is generated by the analyst to external to this object to initiate a game (Event:*Start game*).

Immediate Response Model:

None.

Delayed Response Model:

None.

State:(Starting Game)Immediate Response Model:

If TRUE, then

Set SERVE =(A), Set SHOT=(0), Set SCORE_A=(0), Set SCORE_B=(0), Set WINNER=(Unknown), Generate Event:*Start new rally*.

Delayed Response Model:

None.

State:(Waiting)

The events to transition from this state are generated external to this object. Event:*Player A misses shot* and Event:*Player A faults 2nd serve* are generated by (A). Event:*Player B misses shot* and Event:*Player B faults 2nd serve* are generated by (B).

Immediate Response Model:

None.

Delayed Response Model:

None.

State:(Player A Wins Rally)Immediate Response Model:

If SERVE=(A) and SCORE_A=(14), then

increment SCORE_A, Set WINNER=(A), Generate Event:*End game*.

Else if SERVE =(A), then

increment SCORE_A, Set SHOT=(A), Generate Event:*Start new rally*.

Else Set SHOT=(A), Set SERVE=(A), Generate Event:*Start new rally*.

Delayed Response Model:

None.

State:(Player B Wins Rally)Immediate Response Model:

If SERVE=(B) and SCORE_B=(14), then

increment SCORE_B, Set WINNER=(B), Generate Event:*End game*.

Else if SERVE =(B), then

increment SCORE_B, Set SHOT=(B), Generate Event:*Start new rally*.

Else Set SHOT=(B), Set SERVE=(B), Generate Event:*Start new rally*.

Delayed Response Model:

None.

D. Example Details for Circuit Analysis for Nuclear Power Plant Cable Fires

This appendix provides a description of the system structure, interactions, and state transition diagrams for the circuit analysis for nuclear power plants presented in Section 5.4 of the main body of the report. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams. The circuit of interest for the circuit analysis example is a solenoid-operated valve control circuit (Figure D-1).

Figure D-1. Example solenoid-operated valve control circuit.

D.1 System Structure and Interactions

The system structure diagram for this example problem is shown in Figure D-2. The description of the functional blocks is provided in Table D-1. The interaction diagram is provided in Figure D-3. The functional interactions between the blocks are indicated by directed arrows and are specified in Table D-2.

D.2 Load Object

The Load Object (L) is an element in a partial circuit that represents loads (for example solenoid coils, indicators, motors) that can draw from + or - voltage. Attributes for this object are defined to provide information about the physical connections and electric potential of a load in a partial circuit. The state transition diagram for the load object is shown in Figure D-4. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-2. System structure diagram for the circuit analysis model.

Table D-1. Functional blocks for the circuit analysis model.

L =	Load: an element in a partial circuit that represents loads (for example solenoid coils, indicators, motors); can draw from + or - voltage; can be connected directly between + and - voltage without causing a short circuit.
W =	Conductor (wire): an element in a partial circuit that represents conductors; connects other elements of the circuit with 0 voltage drop in normal operation; if connected directly between + and - voltage sources, a short circuit results.
S =	Switch: an element in a partial circuit that can act as an open circuit or conductor in normal operation.
P _i =	Power Supply: an element in a partial circuit that represents + or - design voltage for Partial Circuit n; for this analysis, the power supply is assumed to provide “enough” current to power the loads described above.
C _n =	Partial Circuit n: a series connection of one or more circuit elements and their possible connection to a power supply.
CC =	Circuit Completer: establishes conditions of connectivity between partial circuits.
CA =	Circuit Analyzer: infers conditions of current flow in a circuit based on conditions in the partial circuits and the corresponding circuit completer.

Figure D-3. Interaction diagram for the circuit analysis model.

Events and State Transitions

This object has only one state - (Load). No state transitions occur; events external to this object will cause changes in object attributes.

Attributes (Values)

CONNECTIVITY (Conducting, Not Conducting)

V-DROP (0, Operating, Open)

Initial Conditions

Analyst must define loads as elements of a partial circuit of interest. Initial conditions are assumed to be normal operation for each respective load defined by the analyst.

CONNECTIVITY=(Conducting); V-DROP=(Operating)

State:(Load)

Immediate Response Model:

None.

Delayed Response Model:

None.

Table D-2. Functional interactions for the circuit analysis model.

L-C _n :	A load provides to its partial circuit information on its physical connections and electric potential drop.
W-C _n :	A conductor provides to its partial circuit information on its physical connections and electric potential drop.
S-C _n :	A switch provides to its partial circuit information on its physical connections and electric potential drop.
P _i -C _n :	Power supply i provides to a partial circuit information about physical connections and electric potential.
C _n -L:	Partial circuit n provides to its loads information about current flow.
C _n -W:	Partial circuit n provides to its conductors information about current flow.
C _n -S:	Partial circuit n provides to its switches information about current flow.
C _n -P _i :	Partial circuit n provides to its power supply information about current flow.
C _n -CA:	Partial circuit n provides to the circuit analyzer information on its physical connections and electric potential.
CA-C _n :	The circuit analyzer provides to the partial circuit information on current flow through the partial circuit.
CA-CC:	The circuit analyzer provides to the circuit completer information on current flow through the conductors between partial circuits.
CC-CA:	The circuit completer provides to the circuit analyzer information on physical connections for the conductors between partial circuits.

Figure D-4. State transition diagram for the Load Object.

D.3 Conductor Object

The Conductor Object (W) is an element in a partial circuit that represents a conductor (wire) that can draw from + or - voltage. Attributes for this object are defined to provide information about the physical connections and electric potential of a conductor in a partial circuit. The state

transition diagram for the conductor object is shown in Figure D-5. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-5. State transition diagram for the Conductor Object.

Events and State Transitions

This object has only one state - (Conduct). No state transitions occur; events external to this object will cause changes in object attributes.

Attributes (Values)

CONNECTIVITY (Conducting, Not Conducting)

V-DROP (0, Open)

Initial Conditions

The analyst must define conductors as elements of a partial circuit of interest. Initial conditions are assumed to be normal operation.

CONNECTIVITY=(Conducting); V-DROP=(0)

State:(Conduct)

Immediate Response Model:

None.

Delayed Response Model:

None.

D.4 Switch Object

The Switch Object (S) is an element in a partial circuit that can act as an open circuit or conductor in normal operation. Attributes for this object are defined to provide information about the physical connections and electric potential of a switch in a partial circuit. The state transition diagram for the switch object is shown in Figure D-6. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-6. State transition diagram for the Switch Object.

Events and State Transitions

Event:*Switch closes* is generated from outside the circuit system, for example, by a human operator, and causes transition from State:(Open) to State:(Closed).

Event:*Switch opens* is generated from outside the circuit system, for example, by a human operator, and causes transition from State:(Closed) to State:(Open).

Attributes (Values)

CONNECTIVITY (Conducting, Not Conducting)

V-DROP (0, Open)

Initial Conditions

Analyst must define switches as elements of a partial circuit of interest. Initial conditions are assumed to be normal operation for an open switch.

CONNECTIVITY=(Not Conducting); V-DROP=(Open)

State:(Open)

Immediate Response Model:

If TRUE, then

Set S:CONNECTIVITY=(Not Conducting) and Set S:V-DROP=(Open)

Delayed Response Model:

None.

State:(Closed)

Immediate Response Model:

If TRUE, then

Set S:CONNECTIVITY=(Conducting) and Set S:V-DROP=(0)

Delayed Response Model:

None.

D.5 Power Supply Object

The Power Supply Object (P_i) is an element in a partial circuit that represents + or - design voltage for Partial Circuit n . For this analysis, the power supply is assumed to provide “enough” current to power the loads (L) in its partial circuit. A circuit model may include one or more power supplies that provide power for one or more partial circuits. Attributes for this object are defined to provide information about the physical connections and polarity of a power supply in a partial circuit. The state transition diagram for the power supply object is shown in Figure D-7. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-7. State transition diagram for the Power Supply Object.

Events and State Transitions

Event:*Fuse blows* is generated by the Partial Circuit Object (C_n) and causes transition from State:(Provides Power) to State:(Does Not Provide Power)

Attributes (Values)

CONNECTIVITY (Conducting, Not Conducting)

POLARITY (+, -)

Initial Conditions

Analyst must define a power supply and its respective polarity as an element of a partial circuit of interest. Initial conditions are assumed to be normal operation for design voltage.

CONNECTIVITY=(Conducting); POLARITY=(+) OR

CONNECTIVITY=(Conducting); POLARITY=(-)

State:Provides Power

Immediate Response Model:

If TRUE, then

Set P_i :CONNECTIVITY=(Conducting)

Delayed Response Model:

None.

State:Does Not Provide Power

Immediate Response Model:

If TRUE, then

Set P_i:CONNECTIVITY=(Not Conducting)

Delayed Response Model:

None.

D.6 Partial Circuit Object

The Partial Circuit Object (C_n) is a series connection of one or more circuit elements and their possible connection to a power supply. Attributes for this object are determined by input from the associated circuit elements and are defined to provide information about the conduction and polarity of the partial circuit. The state transition diagram for the partial circuit object is shown in Figure D-8. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-8. State transition diagram for the Partial Circuit Object.

Events and State Transitions

This object has only one state - (Determine C_n Elements & Attributes). No state transitions occur; events external to this object will cause changes in object attributes.

Attributes (Values)

POLARITY (+, -, Not Determined)

CONDUCTION (Open, Direct, Through Load, Unknown)

Initial Conditions

Initial conditions are set to await information from circuit components.

POLARITY=(Not Determined); CONDUCTION=(Unknown)

State:(Determine C_n Elements and Attributes)

Immediate Response Model:

If CA:CURRENT_n=(Short Circuit) AND P_i:CONNECTIVITY=(Conducting), then

Generate Event:*Fuse blows*

Else If C_n:POLARITY=(Not Determined) AND P_i:POLARITY=(+), then

Set C_n:POLARITY=(+)

Else If C_n:POLARITY=(Not Determined) AND P_i:POLARITY=(-), then

Set C_n:POLARITY=(-)

Else If L:CONNECTIVITY=(Not Conducting) OR W:CONNECTIVITY=(Not Conducting) OR S:CONNECTIVITY=(Not Conducting) OR P_i:CONNECTIVITY=(Not Conducting), then

Set C_n:CONDUCTION=(Open)

Else If L:V-DROP=(Operating), then

Set C_n:CONDUCTION=(Through Load)

Else Set C_n:CONDUCTION=(Direct)

Delayed Response Model:

None.

D.7 Circuit Completer Object

The Circuit Completer Object (CC) establishes all possible conditions of connectivity in a set of partial circuits. The number of partial circuits and paths between them are defined when the model of the circuit of interest is developed. For this example, the circuit of interest is comprised of six partial circuits with three paths (1-2, 3-4, 5-6) between each of two partial circuits. The state transition diagram for the circuit completer object is shown in Figure D-9. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-9. State transition diagram for the Circuit Completer Object.

Events and State Transitions

This object has only one state -(Complete Circuit). No state transitions occur; once this object is initialize, it will progress to generate all possible conditions of completion for the paths between specified partial circuits.

Attributes (Values)

PATH 1-2 (Closed, Open, Short to +, Short to -, Short to Each Other)

PATH 3-4 (Closed, Open, Short to +, Short to -, Short to Each Other)

PATH 5-6 (Closed, Open, Short to +, Short to -, Short to Each Other)

Initial Conditions

Initial conditions for each path are assumed to be normal operation.

PATH 1-2=(Closed)

PATH 3-4=(Closed)

PATH 5-6=(Closed)

State:(Complete Circuit)

Immediate Response Model:

None; we're not changing anything as other outside events happen.

Delayed Response Model:

Always do this:

P=1/13, Delay 10, then Set CC:Path 1-2=(Open) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 1-2=(Short to +) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 1-2=(Short to -) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 3-4=(Open) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 3-4=(Short to +) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 3-4=(Short to -) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 5-6=(Open) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 5-6=(Short to +) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 5-6=(Short to -) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 1-2=(Short to Each Other) AND Set CC:Path3-4=(Short to Each Other) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 1-2=(Short to Each Other) AND set CC:Path5-6=(Short to Each Other) AND Generate Event:*ReSet/New Conditions*

P=1/13, Delay 10, then set CC:Path 3-4=(Short to Each Other) AND Set CC:Path5-6=(Short to Each Other) AND Generate Event:*Reset/New Conditions*

P=1/13, Delay 10, then Set CC:Path 1-2=(Short to Each Other) AND set CC:Path3-4=(Short to Each Other) AND Set CC:Path 5-6=(Short to Each Other) AND Generate Event:*Reset/New Conditions*

D.8 Circuit Analyzer Object

The Circuit Analyzer Object (CA) infers conditions of current flow in a circuit based on conditions in the partial circuits and the corresponding circuit completer. The conditions of current flow are determined for each partial circuit in the circuit of interest. For this example, the circuit of interest is comprised of six partial circuits with three paths (1-2, 3-4, 5-6) between each of two partial circuits. The state transition diagram for the circuit analyzer object is shown in Figure D-10. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure D-10. State transition diagram for the Circuit Analyzer Object.

Events and State Transitions

Event:*Analyze* is generated by the Circuit Analyzer Object (CA) and causes transition from State:(Initialize) to State:(Assess).

Event:*Reset/New Conditions* is generated by the Circuit Completer Object (CC) and causes transition from State:(Assess) to State:(Initialize).

Attributes (Values)

CURRENT_n (Normal, None, Short Circuit, Unknown)

MAXIMUM+CURRENT POTENTIAL (Open, Direct, Through Load, Unknown)

MAXIMUM-CURRENT POTENTIAL (Open, Direct, Through Load, Unknown)

Initial Conditions

Initial conditions are assumed to be normal circuit operation.

CURRENT₁=(Unknown); CURRENT₂=(Unknown); CURRENT₃=(Unknown);

CURRENT₄=(Unknown); CURRENT₅=(Unknown); CURRENT₆=(Unknown);

MAXIMUM+CURRENT POTENTIAL=(Unknown);

MAXIMUM-CURRENT POTENTIAL=(Unknown);

State:(Initialize)Immediate Response Model:

If TRUE, then

Set CURRENT₁=(Unknown) AND Set CURRENT₂=(Unknown) AND Set
CURRENT₃=(Unknown) AND Set CURRENT₄=(Unknown) AND Set
CURRENT₅=(Unknown) AND Set CURRENT₆=(Unknown) AND Set MAXIMUM+
CURRENT POTENTIAL=(Unknown) AND Set MAXIMUM-CURRENT
POTENTIAL=(Unknown) AND Generate Event:*Analyze*

Delayed Response Model:

None.

State:(Assess)Immediate Response Model:

If CC:PATH1-2=(Open) AND CURRENT₁=(Unknown), then

Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:PATH1-2=(Closed) AND C₁:Polarity=(+) AND C₂:Polarity=(+) AND
CURRENT₁=(Unknown), then

Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:PATH1-2=(Closed) AND C₁:Polarity=(-) AND C₂:Polarity=(-) AND
CURRENT₁=(Unknown), then

Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:PATH1-2=(Closed) AND (C₁:Conduction=(Open) OR C₂:Conduction=(Open)) AND
CURRENT₁=(Unknown), then

Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:PATH1-2=(Closed) AND C₁:Conduction=(Through Load) AND
C₂:Conduction=(Through Load) AND CURRENT₁=(Unknown), then

Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:PATH1-2=(Closed) AND CURRENT₁=(Unknown), then

Set CURRENT₁=(Normal) AND Set CURRENT₂=(Normal)

Else If CC:PATH1-2=(Short to +) AND C₁:Conduction=(Direct) AND C₁:Polarity=(-) AND
CURRENT₁=(Unknown), then

Set CURRENT₁=(Short Circuit)

Else If CC:PATH1-2=(Short to +) AND C₂:Conduction=(Direct) AND C₂:Polarity=(-) AND
CURRENT₂=(Unknown), then

Set CURRENT₂=(Short Circuit)

Else If CC:PATH1-2=(Short to +) AND C₁:Conduction=(Through Load) AND C₁:Polarity=(-)
AND CURRENT₁=(Unknown), then

Set CURRENT₁=(Normal)

Else If CC:PATH1-2=(Short to +) AND C₂:Conduction=(Through Load) AND C₂:Polarity=(-)
AND CURRENT₂=(Unknown), then

Set CURRENT₂=(Normal)

Else If CC:PATH1-2=(Short to +) AND CURRENT₁=(Unknown), then

Set CURRENT₁=(None)

Else If CC:Path1-2=(Short to +) AND CURRENT₂=(Unknown), then
Set CURRENT₂=(None)

Else If CC:PATH1-2=(Short to -) AND C₁:Conduction=(Direct) AND C₁:Polarity=(+) AND
CURRENT₁=(Unknown), then
Set CURRENT₁=(Short Circuit)

Else If CC:PATH1-2=(Short to -) AND C₂:Conduction=(Direct) AND C₂:Polarity=(+) AND
CURRENT₂=(Unknown), then
Set CURRENT₂=(Short Circuit)

Else If CC:PATH1-2=(Short to -) AND C₁:Conduction=(Through Load) AND C₁:Polarity=(+)
AND CURRENT₁=(Unknown), then
Set CURRENT₁=(Normal)

Else If CC:PATH1-2=(Short to -) AND C₂:Conduction=(Through Load) AND C₂:Polarity=(+)
AND Set CURRENT₂=(Unknown), then
Set CURRENT₂=(Normal)

Else If CC:PATH1-2=(Short to -) AND CURRENT₁=(Unknown), then
Set CURRENT₁=(None)

Else If CC:Path=(Short to -) AND CURRENT₂=(Unknown), then
Set CURRENT₂=(None)

Else If CC:PATH3-4=(Open) AND CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:PATH3-4=(Closed) AND C₃:Polarity=(+) AND C₄:Polarity=(+) AND
CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:PATH3-4=(Closed) AND C₃:Polarity=(-) AND C₂:Polarity=(-) AND
CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:PATH3-4=(Closed) AND C₃:Conduction=(Open) OR C₄:Conduction=Open AND
CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:PATH3-4=(Closed) AND C₃:Conduction=(Through Load) AND
C₄:Conduction=(Through Load) AND CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:PATH3-4=(Closed) AND CURRENT₃=(Unknown), then
Set CURRENT₃=(Normal) AND Set CURRENT₄=(Normal)

Else If CC:PATH3-4=(Short to +) AND C₃:Conduction=(Direct) AND C₃:Polarity=(-) AND
CURRENT₃=(Unknown), then
Set CURRENT₃=(Short Circuit)

Else If CC:PATH3-4=(Short to +) AND C₄:Conduction=(Direct) AND C₄:Polarity=(-) AND
CURRENT₄=(Unknown), then
Set CURRENT₄=(Short Circuit)

Else If CC:PATH3-4=(Short to +) AND C₃:Conduction=(Through Load) AND
C₃:Polarity=(-) AND CURRENT₃=(Unknown), then
Set CURRENT₃=(Normal)

Else If CC:PATH3-4=(Short to +) AND C₄:Conduction=(Through Load) AND
 C₄:Polarity=(-) AND CURRENT₄=(Unknown), then
 Set CURRENT₄=(Normal)

Else If CC:PATH3-4=(Short to +) AND CURRENT₃=(Unknown), then
 Set CURRENT₃=(None)

Else If CC:PATH3-4=(Short to +) AND CURRENT₄=(Unknown), then
 Set CURRENT₄=(None)

Else If CC:PATH3-4=(Short to -) AND C₃:Conduction=(Direct) AND C₃:Polarity=(+) AND
 CURRENT₃=(Unknown), then
 Set CURRENT₃=(Short Circuit)

Else If CC:PATH3-4=(Short to -) AND C₄:Conduction=(Direct) AND C₄:Polarity=(+) AND
 CURRENT₄=(Unknown), then
 Set CURRENT₄=(Short Circuit)

Else If CC:PATH3-4=(Short to -) AND C₃:Conduction=(Through Load) AND
 C₃:Polarity=(+) AND CURRENT₃=(Unknown), then
 Set CURRENT₃=(Normal)

Else If CC:PATH3-4=(Short to -) AND C₄:Conduction=(Through Load) AND
 C₄:Polarity=(+) AND Set CURRENT₄=(Unknown), then
 Set CURRENT₄=(Normal)

Else If CC:PATH3-4=(Short to -) AND CURRENT₃=(Unknown), then
 Set CURRENT₃=(None)

Else If CC:PATH3-4=(Short to -) AND CURRENT₄=(Unknown), then
 Set CURRENT₄=(None)

Else If CC:PATH5-6=(Open) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:PATH5-6=(Closed) AND C₅:Polarity=(+) AND C₆:Polarity=(+) AND
 CURRENT₅=(Unknown), then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:PATH5-6=(Closed) AND C₅:Polarity=(-) AND C₆:Polarity=(-) AND
 CURRENT₅=(Unknown), then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:PATH5-6=(Closed) AND C₅:CURRENT₆=(Open) OR C₆:Conduction=Open AND
 CURRENT₅=(Unknown), then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:PATH5-6=(Closed) AND C₅:Conduction=(Through Load) AND
 C₂:Conduction=(Through Load) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:PATH5-6=(Closed) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(Normal) AND Set CURRENT₆=(Normal)

Else If CC:PATH5-6=(Short to +) AND C₅:Conduction=(Direct) AND C₅:Polarity=(-) AND
 CURRENT₅=(Unknown), then
 Set CURRENT₅=(Short Circuit)

Else If CC:PATH5-6=(Short to +) AND C₆:Conduction=(Direct) AND C₆:Polarity=(-) AND
 CURRENT₆=(Unknown), then
 Set CURRENT₆=(Short Circuit)

Else If CC:PATH5-6=(Short to +) AND C₅:Conduction=(Through Load) AND
 C₅:Polarity=(-) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(Normal)

Else If CC:PATH5-6=(Short to +) AND C₆:Conduction=(Through Load) AND
 C₆:Polarity=(-) AND CURRENT₆=(Unknown), then
 Set CURRENT₆=(Normal)

Else If CC:PATH5-6=(Short to +) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(None)

Else If CC:PATH5-6=(Short to +) AND CURRENT₆=(Unknown), then
 Set CURRENT₆=(None)

Else If CC:PATH5-6=(Short to -) AND C₅:Conduction=(Direct) AND C₅:Polarity=(+) AND
 CURRENT₅=(Unknown), then
 Set CURRENT₅=(Short Circuit)

Else If CC:PATH5-6=(Short to -) AND C₆:Conduction=(Direct) AND C₆:Polarity=(+) AND
 CURRENT₆=(Unknown), then
 Set CURRENT₆=(Short Circuit)

Else If CC:PATH5-6=(Short to -) AND C₅:Conduction=(Through Load) AND
 C₅:Polarity=(+) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(Normal)

Else If CC:PATH5-6=(Short to -) AND C₆:Conduction=(Through Load) AND
 C₆:Polarity=(+) AND Set CURRENT₆=(Unknown), then
 Set CURRENT₆=(Normal)

Else If CC:PATH5-6=(Short to -) AND CURRENT₅=(Unknown), then
 Set CURRENT₅=(None)

Else If CC:PATH5-6=(Short to -) AND CURRENT₆=(Unknown), then
 Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₁:Polarity=(+) AND C₁:Conduction=(Direct)) OR
 (C₂:Polarity=(+) AND C₂:Conduction=(Direct))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Direct)

Else If CC:Path3-4=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₃:Polarity=(+) AND C₃:Conduction=(Direct)) OR
 (C₄:Polarity=(+) AND C₄:Conduction=(Direct))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Direct)

Else If CC:Path5-6=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₅:Polarity=(+) AND C₅:Conduction=(Direct)) OR
 (C₆:Polarity=(+) AND C₆:Conduction=(Direct))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Direct)

Else If CC:Path1-2=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₁:Polarity=(+) AND C₁:Conduction=(Through Load)) OR (C₂:Polarity=(+) AND C₂:Conduction=(Through Load))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Through Load)

Else If CC:Path3-4=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₃:Polarity=(+) AND C₃:Conduction=(Through Load)) OR (C₄:Polarity=(+) AND C₄:Conduction=(Through Load))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Through Load)

Else If CC:Path5-6=(Short to Each Other) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) AND ((C₅:Polarity=(+) AND C₅:Conduction=(Through Load)) OR (C₆:Polarity=(+) AND C₆:Conduction=(Through Load))) then
 Set MAXIMUM+CURRENT POTENTIAL=(Through Load)

Else If (CC:Path1-2=(Short to Each Other) OR CC:Path3-4=(Short to Each Other) OR
 CC:Path5-6=(Short to Each Other)) AND MAXIMUM+CURRENT
 POTENTIAL=(Unknown) then
 Set MAXIMUM+CURRENT POTENTIAL=(Open)

Else If CC:Path1-2=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₁:Polarity=(-) AND C₁:Conduction=(Direct)) OR
 (C₂:Polarity=(-) AND C₂:Conduction=(Direct))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Direct)

Else If CC:Path3-4=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₃:Polarity=(-) AND C₃:Conduction=(Direct)) OR
 (C₄:Polarity=(-) AND C₄:Conduction=(Direct))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Direct)

Else If CC:Path5-6=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₅:Polarity=(-) AND C₅:Conduction=(Direct)) OR
 (C₆:Polarity=(-) AND C₆:Conduction=(Direct))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Direct)

Else If CC:Path1-2=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₁:Polarity=(-) AND C₁:Conduction=(Through Load))
 OR (C₂:Polarity=(-) AND C₂:Conduction=(Through Load))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Through Load)

Else If CC:Path3-4=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₃:Polarity=(-) AND C₃:Conduction=(Through Load))
 OR (C₄:Polarity=(-) AND C₄:Conduction=(Through Load))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Through Load)

Else If CC:Path5-6=(Short to Each Other) AND MAXIMUM-CURRENT
 POTENTIAL=(Unknown) AND ((C₅:Polarity=(-) AND C₅:Conduction=(Through Load))
 OR (C₆:Polarity=(-) AND C₆:Conduction=(Through Load))) then
 Set MAXIMUM-CURRENT POTENTIAL=(Through Load)

Else If (CC:Path1-2=(Short to Each Other) OR CC:Path3-4=(Short to Each Other) OR CC:Path5-6=(Short to Each Other)) AND MAXIMUM-CURRENT POTENTIAL=(Unknown) then
Set MAXIMUM-CURRENT POTENTIAL=(Open)

Else If CC:Path1-2=(Short to Each Other) AND (MAXIMUM+CURRENT POTENTIAL=(Open) OR MAXIMUM-CURRENT POTENTIAL=(Open)) AND CURRENT₁=(Unknown), then
Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND (MAXIMUM+CURRENT POTENTIAL=(Open) OR MAXIMUM-CURRENT POTENTIAL=(Open)) AND CURRENT₃=(Unknown), then
Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND (MAXIMUM+CURRENT POTENTIAL=(Open) OR MAXIMUM-CURRENT POTENTIAL=(Open)) AND CURRENT₅=(Unknown), then
Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND C₁:CONDUCTION=(Open), then
Set CURRENT₁=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND C₂:CONDUCTION=(Open), then
Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND C₃:CONDUCTION=(Open), then
Set CURRENT₃=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND C₄:CONDUCTION=(Open), then
Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND C₅:CONDUCTION=(Open), then
Set CURRENT₅=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND C₆:CONDUCTION=(Open), then
Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT POTENTIAL=(Direct) AND C₁:Conduction=(Direct) then
Set CURRENT₁=(Short Circuit)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT POTENTIAL=(Direct) AND C₂:Conduction=(Direct) then
Set CURRENT₂=(Short Circuit)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct) AND C₃:Conduction=(Direct) then
 Set CURRENT₃=(Short Circuit)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct) AND C₄:Conduction=(Direct) then
 Set CURRENT₄=(Short Circuit)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct) AND C₅:Conduction=(Direct) then
 Set CURRENT₅=(Short Circuit)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct) AND C₆:Conduction=(Direct) then
 Set CURRENT₆=(Short Circuit)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₁=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₃=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₅=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND MAXIMUM-CURRENT
 POTENTIAL=(Direct), then
 Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Through Load) AND MAXIMUM-CURRENT
 POTENTIAL=(Through Load) AND C₁:Conduction=(Open) then
 Set CURRENT₁=(None) AND Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Through Load) AND MAXIMUM-CURRENT
 POTENTIAL=(Through Load) AND C₃:Conduction=(Open) then
 Set CURRENT₃=(None) AND Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Through Load) AND MAXIMUM-CURRENT
 POTENTIAL=(Through Load) AND C₅:Conduction=(Open) then
 Set CURRENT₅=(None) AND Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₁:Polarity=(+) AND
 C₁:Conduction=(Direct) then
 Set CURRENT₁=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₂:Polarity=(+) AND
 C₂:Conduction=(Direct) then
 Set CURRENT₂=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₁:Polarity=(-) then
 Set CURRENT₁=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₂:Polarity=(-) then
 Set CURRENT₂=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₁:Polarity=(+), then
 Set CURRENT₁=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₂:Polarity=(+), then
 Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₃:Polarity=(+) AND
 C₃:Conduction=(Direct) then
 Set CURRENT₃=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₄:Polarity=(+) AND
 C₄:Conduction=(Open) then
 Set CURRENT₄=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₃:Polarity=(-) then
 Set CURRENT₃=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₄:Polarity=(-) then
 Set CURRENT₄=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₃:Polarity=(+), then
 Set CURRENT₃=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₄:Polarity=(+), then
 Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₅:Polarity=(+) AND
 C₅:Conduction=(Direct), then
 Set CURRENT₅=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₆:Polarity=(+) AND
 C₆:Conduction=(Open), then
 Set CURRENT₆=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₅:Polarity=(-), then
 Set CURRENT₅=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₆:Polarity=(-), then
 Set CURRENT₆=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₅:Polarity=(+), then
 Set CURRENT₅=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND
 MAXIMUM+CURRENT POTENTIAL=(Direct) AND C₆:Polarity=(+), then
 Set CURRENT₆=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) C₁:Polarity=(-) AND
 C₁:Conduction=(Direct), then
 Set CURRENT₁=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND C₂:Polarity=(-)
 AND C₂:Conduction=(Direct), then
 Set CURRENT₂=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND C₁:Polarity=(+),
 then
 Set CURRENT₁=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND C₂:Polarity=(+),
 then
 Set CURRENT₂=(Normal)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₁=(Unknown) AND C₁:Polarity=(-),
 then
 Set CURRENT₁=(None)

Else If CC:Path1-2=(Short to Each Other) AND CURRENT₂=(Unknown) AND C₂:Polarity=(-),
 then
 Set CURRENT₂=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND C₃:Polarity=(-)
 AND C₃:Conduction=(Direct), then
 Set CURRENT₃=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND C₄:Polarity=(-)
 AND C₄:Conduction=(Open), then
 Set CURRENT₄=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND C₃:Polarity=(+),
 then
 Set CURRENT₃=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND C₄:Polarity=(+),
 then
 Set CURRENT₄=(Normal)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₃=(Unknown) AND C₃:Polarity=(-),
 then
 Set CURRENT₃=(None)

Else If CC:Path3-4=(Short to Each Other) AND CURRENT₄=(Unknown) AND C₄:Polarity=(-),
 then
 Set CURRENT₄=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND C₅:Polarity=(-)
 AND C₅:Conduction=(Direct), then
 Set CURRENT₅=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND C₆:Polarity=(-)
 AND C₆:Conduction=(Open), then
 Set CURRENT₆=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND C₅:Polarity=(+),
 then
 Set CURRENT₅=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND C₆:Polarity=(+),
 then
 Set CURRENT₆=(Normal)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₅=(Unknown) AND C₅:Polarity=(-),
 then
 Set CURRENT₅=(None)

Else If CC:Path5-6=(Short to Each Other) AND CURRENT₆=(Unknown) AND C₆:Polarity=(-),
 then
 Set CURRENT₆=(None)

This Page Intentionally Blank

E. Example Details for Basic Infrastructure Problem

This appendix provides a description of the system structure, interactions, and state transition diagrams for the basic infrastructure problem presented in Section 5.5 of the main body of the report. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams.

E.1 System Structure and Interactions

This example problem is formulated to allow the analyst to consider portions of three separate situations: where a person has neglected all preparation for infrastructure disruption, where a person has prepared by purchasing a portable electric generator, and where a person has made preparations to survive without electricity. The system structure diagrams and interaction lists for these three situations are found in Figure E-1, Figure E-2, and Figure E-3, respectively. The system structure diagram and interactions list for the combined model are found in Figure E-4, while a more complete functional interactions diagram for this combined model is shown in Figure E-5, while the functional interactions are described in Table E-1.

Figure E-1. Components and interactions in the "Unprepared" situation.

Figure E-2. Components and interactions in the "With Generator" situation.

Figure E-3. Components and interactions for the "No Electricity" situation.

Figure E-4. Components and interactions for the combined model.

Figure E-5. Functional interaction diagram for the combined model.

Table E-1. Functional interactions for the combined model.

Com. Gas – Heat	Commercial Gas provides fuel to Heat
Com. Gas – Cooking	Commercial Gas provides fuel to Cooking
Com. Power – Electricity	Commercial Power provides electric power to Electricity
Generator – Electricity	Generator provides electric power to Electricity (Demand for Electricity passed back)
Electricity – Light	Electricity provides power to provide Light (Demand for Electricity passed back)
Electricity – Cooking	Electricity provides power to enable Cooking (Demand for Electricity passed back)
Electricity – Heat	Electricity provides power to provide Heat (Demand for Electricity passed back)
Electricity – Water	Electricity provides power to pump Water from the well (Demand for Electricity passed back)
Batteries – Light	Batteries provide power to provide Light (Demand for Batteries resources passed back)
Candles – Light	Candles provide energy to provide Light (Demand for Candles resources passed back)
Matches – Light	Matches provide fire that enables candles or lanterns to provide Light (Demand for Matches resources passed back)
Matches – Cooking	Matches provide fire that enables camp stove to provide Cooking (Demand for Matches resources passed back)
Matches – Heat	Matches provide fire that enables firewood or kerosene heater to provide Heat (Demand for Matches resources passed back)
Cooking – Water	Cooking provides boiling to purify Water
Bottles – Water	Bottles provide reserves to Water (Demand for water resources passed back)
Kerosene – Heat	Kerosene provides fuel to Heat (Demand for Kerosene resources passed back)
Firewood – Heat	Firewood provides fuel to Heat (Demand for Firewood resources passed back)
Water – Camp Fuel	Water provides an (additional) energy demand to Camp Fuel
Camp Fuel – Light	Camp Fuel provides energy to provide Light (Demand for Camp Fuel is passed back)
Camp Fuel – Cooking	Camp Fuel provides energy to provide Cooking (Demand for Camp Fuel is passed back)

E.2 The Vital Household Functions Objects

The first four objects to be discussed are the four vital household functions that are to be maintained during infrastructure disruptions: the Light Object, the Heat Object, the Cooking Object, and the Water Object. These objects are similar to one another in that each functions to track the various sources that can provide its vital function and ultimately determines whether or not that function is being met. The complete OBEST models for these four objects are found in Figure E-6, Figure E-7, Figure E-8, and Figure E-9, respectively. The state transition diagram within each of these figures provides the object's states and events, while the text within the figure provides the object's attributes and "if – then – else if – else" scripts and logical conditions for each model and state. Each object's initial conditions follow its figure.

Figure E-6. States, attributes, and response models for the Light Object.

The initial conditions for the Light Object are State:(Lights) and attribute value Source=(Electric).

Figure E-7. States, attributes, and response models for the Heat Object.

The initial conditions for the Heat Object are State:(Warm) and attribute value Source=(Furnace).

Figure E-8. States, attributes, and response models for the Cooking Object.

The initial conditions for the Cooking Object are State:(Cooking) and attribute value Source=(Stove).

Figure E-9. States, attributes, and response models for the Water Object.

The initial conditions for the Water Object are State:(Consuming) and attribute value Source=(Well).

E.3 The Consumable Supplies Objects

The following eight objects in the infrastructure disruption preparation model represent consumable supplies that would be used in the event of a major infrastructure outage. These objects are all extremely similar in their structure in behavior. Each has one attribute named Supply that indicates whether the reserve supply of this commodity remains available or has been exhausted. These two discrete values for the attribute correspond to the two states that make up each of these object models – again, one to indicate that this commodity remains available and one to indicate that it is unavailable. Each object's IRM is used to ensure the proper setting of the Supply attribute and to manipulate the object's internal clock. The clock represents a sort of "countdown" until this object's resources will be exhausted. The delay that is built into each object's DRM indicates the number of days the analyst expects this resource to last when used in a particular fashion. The object's clock is started when the appropriate vital functions object indicates that this object's supplies are being used as a source to fulfill that vital function's needs. When the delay time is exhausted, this object's resources are assumed to be fully depleted, so the DRM generates an event that will cause the object to transition into its depleted or unavailable state. The DRM may include multiple delay times to indicate that different usage rates occur when the resource is used in different ways.

The complete object model for each of these eight objects is presented in a figure on the following pages. The state transition diagram within each of these figures provides the object's states and events, while the text within the figure provides the object's attributes and "if – then – else if – else" scripts and logical conditions for each model and state. Each object's initial conditions follow its figure.

The object models and figures are as follows:

- The Camp Fuel Object – Figure E-10
- The Bottles Object – Figure E-11
- The Batteries Object – Figure E-12
- The Candles Object – Figure E-13
- The Kerosene Object – Figure E-14
- The Firewood Object – Figure E-15
- The Matches Object – Figure E-16, and
- The Generator Object – Figure E-17

Figure E-10. States, attributes, and response models for the Camp Fuel Object.

The initial conditions for the Camp Fuel Object are State:(Fuel) and attribute value Supply=(Fuel).

Figure E-11. States, attributes, and response models for the Bottles Object.

The initial conditions for the Bottles Object are State:(Water) and attribute value Supply=(Water).

Figure E-12. States, attributes, and response models for the Batteries Object.

The initial conditions for the Batteries Object are State:(Power) and attribute value Supply=(Batteries).

Figure E-13. States, attributes, and response models for the Candles Object.

The initial conditions for the Candles Object are State:(Candles) and attribute value Supply=(Candles).

Figure E-14. States, attributes, and response models for the Kerosene Object.

The initial conditions for the Kerosene Object are State:(Kerosene) and attribute value Supply=(Kerosene).

Figure E-15. States, attributes, and response models for the Firewood Object.

The initial conditions for the Firewood Object are State:(Firewood) and attribute value Supply=(Firewood).

Figure E-16. States, attributes, and response models for the Matches Object.

The initial conditions for the Matches Object are State:(Matches) and attribute value Supply=(Matches).

Figure E-17. States, attributes, and response models for the Generator Object.

The initial conditions for the Generator Object are State:(Available) and attribute value Supply=(Available).

E.4 The Commercial Power and Commercial Gas Objects

The Electricity Object summarizes for the remainder of the object model the current status of electric power and its source. This functionality is implemented in a single state with an IRM and no DRM, as seen in Figure E-18. It uses only one state and no events (as seen in the state transition diagram in Figure E-18). Its attributes, values and response models are shown in the text in the figure. Its initial conditions are State:(Powering), and attribute values Source=(Commercial) and Power=(Power).

Figure E-18. States, attributes, and response models for the Electricity Object.

The remaining two objects in this OBEST model model the availability of commercial electric power and commercial natural gas. These objects are probabilistic in nature, and are designed to

provide the remainder of the OBEST model with a variety of "boundary conditions" that represent the array of possible infrastructure outages that might have been expected as a result of the transition to the year 2000. The state transition diagram for the Commercial Electric Power Object is found in Figure E-19. This diagram shows the states and events for this object. The attributes, initial conditions and response models (with their "if – then – else if – else" logic) and scripts follow the figure.

Figure E-19. States transition diagram for the Commercial Power Object.

Object Attributes (Values)

Supply (Available, Restored, None)

Initial Conditions

The initial conditions are State:(Available) and attribute value Supply=(Available).

State:(Available)

Immediate Response Model:

If Supply=(None) then

Set Supply=(Restored)

Else If Supply=(Available) then

P=0.1, Set Supply=(Restored)

P=0.9, Reset My Clock, Generate Event:*Power's Out*

Else

Do Nothing

Delayed Response Model:

None.

State:(Dead)**Immediate Response Model:**

If Supply=(Available) then
Set Supply=(None)

Delayed Response Model:

Always do this:

P=0.1, Delay 0.5, Generate Event:*Power's On*

P=0.1, Delay 1, Generate Event:*Power's On*

P=0.1, Delay 2, Generate Event:*Power's On*

P=0.1, Delay 4, Generate Event:*Power's On*

P=0.1, Delay 7, Generate Event:*Power's On*

P=0.1, Delay 10, Generate Event:*Power's On*

P=0.1, Delay 15, Generate Event:*Power's On*

P=0.1, Delay 20, Generate Event:*Power's On*

P=0.1, Delay 25, Generate Event:*Power's On*

P=0.1, Delay 30, Generate Event:*Power's On*

The state transition diagram for the Commercial Gas Object is found in Figure E-20. This diagram shows the states and events for this object. The attributes, initial conditions and response models (with their “if – then – else if – else” logic) and scripts follow the figure.

Figure E-20. State transition diagram for the Commercial Gas object.

Object Attributes (Values)

Supply (Available, Restored, None)

Initial Conditions

The initial conditions are State:(Available) and attribute value Supply=(Available).

State:(Available)

Immediate Response Model:

If Supply=(None) then
 Set Supply=(Restored)
Else If Supply=(Available) then
 P=0.1, Set Supply=(Restored)
 P=0.9, Reset My Clock, Generate Event:*Gas Out*
Else
 Do Nothing

Delayed Response Model:

None.

State:(Dead)

Immediate Response Model:

If Supply=(Available) then
 Set Supply=(None)

Delayed Response Model:

Always do this:

P=0.1, Delay 1, Generate Event:*Gas On*
P=0.1, Delay 2, Generate Event:*Gas On*
P=0.1, Delay 3, Generate Event:*Gas On*
P=0.1, Delay 4, Generate Event:*Gas On*
P=0.1, Delay 6, Generate Event:*Gas On*
P=0.1, Delay 8, Generate Event:*Gas On*
P=0.1, Delay 10, Generate Event:*Gas On*
P=0.1, Delay 15, Generate Event:*Gas On*
P=0.1, Delay 25, Generate Event:*Gas On*
P=0.1, Delay 40, Generate Event:*Gas On*

This Page Intentionally Blank

F. Example Details for Aircraft Spacing for Runway Operations

This appendix provides a description of the system structure, interactions, and state transition diagrams for the aircraft spacing for runway operations example presented in Section 5.6 of the main body of the report. The descriptions of the airport/airspace, taxi aircraft and landing aircraft include a discussion of their respective state transition diagrams. This description includes the sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models of the state transition diagrams.

F.1 System Structure and Interactions

The system structure diagram for this example problem is shown in Figure F-1. The description of the functional blocks is provided in Table F-1. The interaction diagram is provided in Figure F-2. The functional interactions between the blocks are indicated by directed arrows and are specified in Table F-2.

Figure F-1. System structure diagram model of runway operations.

Table F-1. Functional blocks for runway operations.

AP =	Airport/Airspace: Represents the physical layout of the airport/airspace and tracks aircraft location within this layout.
TC =	Tower Controller: Provides commands and responses and monitors airport/airspace and aircraft.
TA =	Taxi Aircraft: An aircraft either taxing from the gate to the runway for takeoff or from the runway to the gate after landing.
LA =	Landing Aircraft: An aircraft either flying in for landing on a runway or taking off.
Env =	Environment: Provides environmental context for an analysis, including weather conditions, visibility, air traffic levels, flight crew performance, lighting, radio conditions, and airport signage.

Figure F-2. Interaction diagram for the model of runway operations.

Table F-2. Functional interactions for runway operations.

AP-TC:	The airport/airspace provides to the tower controller information on aircraft positions.
TC-AP:	The tower controller provides to the airport/airspace information about runway reservations.
TC-TA:	The tower controller provides taxi commands to a taxi aircraft and responds to requests from a taxi aircraft.
TA-AP:	A taxi aircraft provides to the airport/airspace information about its position.
TA-TC:	A taxi aircraft makes requests to the tower controller and responds to commands from the tower controller.
LA-AP:	A landing aircraft provides to the airport/airspace information about its position.
Env-TC:	The environment provides to the tower controller information about distractions, visibility, and radio interference.
Env-TA:	The environment provides to a taxi aircraft information about distractions, visibility, and radio interference.
Env-LA:	The environment provides to a landing aircraft information about distractions, visibility, and radio interference.

F.2 Airport/Airspace Object

The Airport/Airspace Object (AP) provides a representation the physical layout of the airport/airspace and tracks aircraft location within this layout. Attributes for this object are defined to provide information about the physical locations of taxi and landing aircraft and about the status of the runway. The physical runway/taxiway layout used for this example is provided in Figure F-3. The state transition diagram for the airport/airspace object is shown in Figure F-4. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

This object has only one state - (Status).

No state transitions; events external to this object will cause changes in object attributes.

Attributes (Values)

Taxi Aircraft Location – TAL (G, T12, T3, X, T5, BT)

Landing Aircraft Locations – LAL (A, R1, R23, X, R5, BR)

Runway Status – RS (Reserved, Available)

Initial Conditions

TAL=(G)

LAL=(A)

RS=(Reserved)

State:(Airport Status)Immediate Response Model:

If TA:Location=(G) AND TAL/=(G) then

Set TAL=(G), Generate Event:TA_New_Location

Else If TA:Location=(T12) AND TAL/=(T12) then

Set TAL=(T12), Generate Event:TA_New_Location

Else If TA:Location=(T3) AND TAL/=(T3) then

Set TAL=(T3), Generate Event:TA_New_Location

Else If TA:Location=(X) AND TAL/=(X) then

Set TAL=(X), Generate Event:TA_New_Location

Else If TA:Location=(T5) AND TAL/=(T5) then

Set TAL=(T5), Generate Event:TA_New_Location

Else If TA:Location=(BT) AND TAL/=(BT) then

Set TAL=(BT) Generate Event:TA_New_Location

Else If LA:Location=(A) AND LAL/=(A) then

Set LAL=(A), Generate Event:LA_New_Location

Else If LA:Location=(R1) AND LAL/=(R1) then

Set LAL=(R1), Generate Event:LA_New_Location

Else If LA:Location=(R23) AND LAL/=(R23) then

Set LAL=(R23), Generate Event:LA_New_Location

Else If LA:Location=(X) AND LAL/=(X) then

Set LAL=(X), Generate Event:LA_New_Location

Else If LA:Location=(R5) AND LAL/=(R5) then

Set LAL=(R5), Generate Event:LA_New_Location

Else If LA:Location=(BR) AND LAL/=(BR) then

Set LAL=(BR), Generate Event:LA_New_Location

Else If TC:Land_Clear=(CTL) AND RS/=(Reserved) then

Set RS=(Reserved)

Else If TC:Land_Clear=(NO) AND RS/=(Available) then

Set RS=(Available)

Else Do Nothing

Delayed Response Model:

None.

Figure F-3. Physical layout for the Airport/Airspace Object.

Figure F-4. State transition diagram for the Airport/Airspace Object.

F.3 Landing Aircraft Object

The Landing Aircraft Object (AP) represents an aircraft either flying in for landing on a runway or taking off. Attributes for this object are defined to provide information about the physical location and motion of the landing aircraft and its mode of operation (landing or takeoff). The state transition diagram for the landing aircraft object is shown in Figure F-5. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Figure F-5. State transition diagram for the Landing Aircraft Object.

Events and State Transitions

Event:*LA_New_Location (LA-NL)* is generated by the Landing Aircraft Object and also by the Airport/Airspace Object and causes the following transitions:

- from State:(Flying) to State:(Change Location)
- from State:(Moving) to State:(Change Location)
- from State:(Stopped) to State:(Change Location)

Event:*LA-Fly* is generated by the Landing Aircraft Object and causes transition from State:(Change Location) to State:(Flying).

Event:*LA-Stop* is generated by the Landing Aircraft Object and causes transition from State:(Change Location) to State:(Stopped).

Event:*LA-Move* is generated by the Landing Aircraft Object and causes transition from State:(Change Location) to State:(Moving).

Attributes (Values)

Mode (Takeoff, Landing)

Location (A, R1, R23, X, R5, BR)

Motion (Stopped, Moving, Flying)

Initial Conditions

Mode=(Landing)

Location=(A)

Motion=(Landing)

State:(Flying)Immediate Response Model:

If Location $\neq$ (A) then

 Set Location=(A)

Else If Motion $\neq$ (Flying) then

 Set Motion=(Flying)

Else Do Nothing

Delayed Response Model: *{This models final approach timing.}*

If Mode=(Takeoff) then

 Terminate Scenario

Else P=0.25, Delay 120, Generate Event:LA_New_Location

 P=0.50, Delay 135, Generate Event:LA_New_Location

 P=0.25, Delay 150, Generate Event:LA_New_Location

State:(Change Location)Immediate Response Model:

If Mode=(Landing) AND Location=(A) then

 Set Location=(R1), Generate Event:LA-Move

Else If Mode=(Landing) AND Location=(R1) then

 Set Location=(R23), Generate Event:LA-Move

Else If Mode=(Landing) AND Location=(R23) then

 Set Location=(X), Generate Event:LA-Move

Else If Mode=(Landing) AND Location=(RX) then

 Set Location=(R5), Generate Event:LA-Move

Else If Mode=(Landing) AND Location=(R5) then

 Set Location=(BR), Set Motion=(Stopped), Generate Event:LA-Stop

Else If (when Mode=(Takeoff) AND) Location=(BR) then

 Set Location=(R5), Generate Event:LA-Move

Else If (when Mode=(Takeoff) AND) Location=(R5) then

 Set Location=(X), Generate Event:LA-Move

Else If (when Mode=(Takeoff) AND) Location=(X) then

 Set Location=(R32), Generate Event:LA-Move

Else If (when Mode=(Takeoff) AND) Location=(R32) then

Set Location=(R1), Generate Event:*LA-Move*
Else (when Mode=(Takeoff AND Location=(R1))
Set Location=(A), Set Motion=(Flying), Generate Event:*LA-Fly*

Delayed Response Model:
None

State:(Moving)

Immediate Response Model:

If Motion!=(Flying) then
Set Motion=(Moving)
Else Do Nothing

Delayed Response Model: {Note: can't be in location A while not flying.}

If Mode=(Landing) AND Location=(R1) then
P=0.25, Delay 7, Generate Event:*LA_New_Location*
P=0.50, Delay 10, Generate Event:*LA_New_Location*
P=0.25, Delay 13, Generate Event:*LA_New_Location*

Else If Mode=(Landing) AND Location=(R23) then
P=0.25, Delay 4, Generate Event:*LA_New_Location*
P=0.50, Delay 5, Generate Event:*LA_New_Location*
P=0.25, Delay 7, Generate Event:*LA_New_Location*

Else If Mode=(Landing) AND Location=(X) then
Delay 3, generate Event:*LA_New_Location*

Else If Mode=(Landing) AND Location=(R5) then
P=0.25, Delay 10, Generate Event:*LA_New_Location*
P=0.50, Delay 15, Generate Event:*LA_New_Location*
P=0.25, Delay 20, Generate Event:*LA_New_Location*

{Note: from here on, mode can only be (Takeoff). Can't have BR while plane is moving.}

Else If Location=(R5) then
P=0.25, Delay 10, Generate Event:*LA_New_Location*
P=0.50, Delay 20, Generate Event:*LA_New_Location*
P=0.25, Delay 30, Generate Event:*LA_New_Location*

Else If Location=(X) then
Delay 3, Generate Event:*LA_New_Location*

Else If Location=(R23) then
P=0.25, Delay 5, Generate Event:*LA_New_Location*
P=0.50, Delay 7, Generate Event:*LA_New_Location*
P=0.25, Delay 9, Generate Event:*LA_New_Location*

Else (*Location*=(*R1*))

P=0.25, Delay 10, Generate Event:*LA_New_Location*

P=0.50, Delay 13, Generate Event:*LA_New_Location*

P=0.25, Delay 16, Generate Event:*LA_New_Location*

{Note that the location cannot be A while in the state Moving – only when Flying.}

State:(Stopped)

Immediate Response Model:

If *Location*!=(*BR*) then

Set *Location*=(*BR*)

Else If *Motion*!=(*Stopped*) then

Set *Motion*=(*Stopped*)

Else Do Nothing

Delayed Response Model: *{This represents the delay before takeoff rolls starts.}*

If *Mode*=(*Landing*) then

Terminate Scenario

Else P=0.25, Delay 15, Generate Event:*LA_New_Location*

F.4 Taxi Aircraft Object

The Taxi Aircraft Object (AP) represents an aircraft either taxiing from the gate to the runway for takeoff or from the runway to the gate after landing. Attributes for this object are defined to provide information about the physical location and motion of the landing aircraft, its mode of operation (landing or takeoff), and pilot response to hold commands from the tower controller. This object initially ignores the tower controller and the environment. The state transition diagram for the taxi aircraft object is shown in Figure F-6. The events and state transitions, attributes, initial conditions, and sets of “if – then – else if – else” scripts and logical conditions for the IRM and DRM models for each state are presented in the following sections.

Events and State Transitions

Event:*TA_New_Location (LA-NL)* is generated by the Taxi Aircraft Object and also by the Airport/Airspace Object and causes the following transitions:

from State:(*Gate Stop*) to State:(*Change Location*)

from State:(*Taxi*) to State:(*Change Location*)

from State:(*Hold*) to State:(*Change Location*)

from State:(*Beyond Model*) to State:(*Change Location*)

Event:*TA-Gate* is generated by the Taxi Aircraft Object and causes transition from State:(*Change Location*) to State:(*Gate Stop*).

Event:*TA-Taxi* is generated by the Taxi Aircraft Object and causes transition the following transitions:

from State:(*Change Location*) to State:(*Taxi*).

from State:(*Hold*) to State:(*Taxi*).

Figure F-6. State transition diagram for the Taxi Aircraft Object.

Event: *TA-Hold* is generated by the Taxi Aircraft Object and causes transition the following transitions:

from State:(Change Location) to State:(Hold)

from State:(Taxi) to State:(Hold)

Event: *TA-Beyond_Model* is generated by the Taxi Aircraft Object and causes transition from State:(Change Location) to State:(Beyond Model).

Attributes (Values)

Mode (Takeoff, Landing)

Location (G, T12, T3, X, T5, BT)

PilotHold (Unset, WillHold, NoHold)

Motion (Gate, Taxi, Hold, Beyond)

Initial Conditions

Mode=(Takeoff)

Location=(G)

PilotHold=(Unset)

Motion=(Gate)

State:(Gate Stop)

Immediate Response Model:

If Location!=(G) then

Set Location=(G)

Else If Motion!=(Gate) then

Set Motion=(Gate)

Else If Mode=(Takeoff) AND PilotHold=(Unset) then
 P=0.999, Set PilotHold=(WillHold)
 P=0.001, Set PilotHold=(NoHold)
 Else Do Nothing

Delayed Response Model: *{This models delays leaving the gate.}*

If Mode=(Landing) then
 Terminate Scenario
 Else P=0.25, Delay 100, Generate Event:TA_New_Location
 P=0.50, Delay 120, Generate Event:TA_New_Location
 P=0.25, Delay 140, Generate Event:TA_New_Location

State:(Change Location)

Immediate Response Model:

If Mode=(Takeoff) AND Location=(G) then
 Set Location=(T12), Generate Event:TA-Taxi
 Else If Mode=(Takeoff) AND Location=(T12) then
 Set Location=(T3), Generate Event:TA-Taxi
 Else If Mode=(Takeoff) AND Location=(T3) then
 Set Location=(X), Generate Event:TA-Taxi
 Else If Mode=(Takeoff) AND Location=(X) then
 Set Location=(T5), Generate Event:TA-Taxi
 Else If Mode=(Takeoff) AND Location=(T5) then
 Set Location=(BT), Set Motion=(Beyond), Generate Event:TA-Beyond Model

 Else If (when Mode=(Landing) AND) Location=(BT) then
 Set Location=(T5), Generate Event:TA-Taxi
 Else If (when Mode=(Landing) AND) Location=(T5) then
 Set Location=(X), Generate Event:TA-Taxi
 Else If (when Mode=(Landing) AND) Location=(X) then
 Set Location=(T3), Generate Event:TA-Taxi
 Else If (when Mode=(Landing) AND) Location=(T3) then
 Set Location=(T12), Generate Event:TA-Taxi
 Else (when Mode=(Landing) AND Location=(T12))
 Set Location=(G), Motion=(Gate), Generate Event:TA-Gate

Delayed Response Model:

None

State:(Taxi)

Immediate Response Model:

If Motion!=(Taxi) then
 Set Motion=(Taxi)
 Else Do Nothing

Delayed Response Model: *{Can't be in location G in Taxi state.}*

If Mode=(Takeoff) AND Location=(T12) then

 P=0.25, Delay 30, Generate Event:TA_New_Location

 P=0.50, Delay 40, Generate Event:TA_New_Location

 P=0.25, Delay 50, Generate Event:TA_New_Location

Else If Mode=(Takeoff) AND Location=(T3) AND PilotHold=(NoHold) then

 P=0.25, Delay 10, Generate Event:TA_New_Location

 P=0.50, Delay 15, Generate Event:TA_New_Location

 P=0.25, Delay 20, Generate Event:TA_New_Location

Else If Mode=(Takeoff) AND Location=(T3) (AND PilotHold=(WillHold)) then

 P=0.25, Delay 10, Generate Event:TA_Hold

 P=0.50, Delay 15, Generate Event:TA_Hold

 P=0.25, Delay 20, Generate Event:TA_Hold

Else If Mode=(Takeoff) AND Location=(X) then

 Delay 10, Generate Event:TA_New_Location

Else If Mode=(Takeoff) (AND Location=(T5)) then

 P=0.25, Delay 15, Generate Event:TA_New_Location

 P=0.50, Delay 23, Generate Event:TA_New_Location

 P=0.25, Delay 30, Generate Event:TA_New_Location

{Note: from here on, mode can only be (Landing). Can't have BT while plane is in Taxi state.}

Else If (Mode=(Landing) AND) Location=(T5) AND PilotHold=(NoHold) then

 P=0.25, Delay 15, Generate Event:TA_New_Location

 P=0.50, Delay 23, Generate Event:TA_New_Location

 P=0.25, Delay 30, Generate Event:TA_New_Location

Else If (Mode=(Landing AND) Location=(T5 (AND PilotHold=(WillHold)) then

 P=0.25, Delay 15, Generate Event:TA_Hold

 P=0.50, Delay 23, Generate Event:TA_Hold

 P=0.25, Delay 30, Generate Event:TA_Hold

Else If (Mode=(Landing) AND) Location=(X) then

 Delay 10, Generate Event:TA_New_Location

Else If (Mode=(Landing) AND) Location=(T3) then

 P=0.25, Delay 10, Generate Event:TA_New_Location

 P=0.50, Delay 15, Generate Event:TA_New_Location

 P=0.25, Delay 20, Generate Event:TA_New_Location

Else (If Mode=(Landing) AND Location=(T12)) then
 P=0.25, Delay 30, Generate Event:TA_New_Location
 P=0.50, Delay 40, Generate Event:TA_New_Location
 P=0.25, Delay 50, Generate Event:TA_New_Location

{Note that the location cannot be G or BT while in the state Taxi.}

State: Hold

{This is a very crude model – they wait 3 minutes until this plane is gone, then proceed. This state's final behavior will need to interface with the tower controller object.}

Immediate Response Model:

None

Delayed Response Model:

Always do this:

 Delay 180, Set PilotHold=(NoHold), Generate Event:TA_New_Location

State:(Beyond Model)

Immediate Response Model:

If Location!=(BT) then

 Set Location=(BT)

Else If Motion!=(Beyond) then

 Set Motion=(Beyond)

Else If Mode=(Landing) AND PilotHold=(Unset) then

 P=0.999, Set PilotHold=(WillHold)

 P=0.001, Set PilotHold=(NoHold)

Else Do Nothing

Delayed Response Model: *{This represents the delay before plane enters taxiway T5.}*

If Mode=(Takeoff) then

 Terminate Scenario

Else P=0.25, Delay 15, Generate Event:TA_New_Location

 P=0.50, Delay 20, Generate Event:TA_New_Location

 P=0.25, Delay 25, Generate Event:TA_New_Location

This Page Intentionally Blank

DISTRIBUTION

External Distribution

Mark A. Cunningham
Probabilistic Risk Analysis Branch
US Nuclear Regulatory Commission
MS TWF 10 E50
Washington, DC 20555-0001

Mary T. Drouin
Probabilistic Risk Analysis Branch
US Nuclear Regulatory Commission
MS TWF 10 E50
Washington, DC 20555-0001

Alan S. Kuritzky
Probabilistic Risk Analysis Branch
US Nuclear Regulatory Commission
MS TWF 10 E50
Washington, DC 20555-0001

Alan M. Rubin
Probabilistic Risk Analysis Branch
US Nuclear Regulatory Commission
MS TWF 10 E50
Washington, DC 20555-0001

Harry F. Martz
Group A1, F-600
Los Alamos National Laboratory
Los Alamos, NM 87545

George Apostolakis
Massachusetts Institute of Technology
Nuclear Engineering Department
77 Massachusetts Avenue, Room 24-223
Cambridge, MA 02139-4307

Kevin Corker
San Jose State University Foundation
NASA Ames Research Center
Mail Stop 262-1
Moffett Field, CA 94035-1000

Irv Statler
APMS Project Manager
NASA Ames Research Center
Code IHS, Mail Stop 262-7
Moffett Field, CA 94035-1000

Mary Connor
NASA Ames Research Center
Code IHS, Mail Stop 262-4
Moffett Field, CA 94035-1000

Mike Abkin
ATAC Corporation
757 N. Mary Avenue
Sunnyvale, CA 94085

Michael Stamatelatos
NASA Headquarters
Code QE
300 E St. SW
Washington, DC 20024-3210

Gareth W. Parry
Division of Systems Safety and Analysis
MS 10 A1
US Nuclear Regulatory Commission
Washington, DC 20555-0001

Nathan O. Siu
Probabilistic Risk Analysis Branch
MS 10 E50
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Sandia Distribution

MS0188	D. L. Chavez, 1030	MS0748	W. Cheng, 6413
MS0405	T. R. Jones, 12333	MS0748	J. A. Forester, 6413
MS0405	M. P. Bohn, 12333	MS0748	S. P. Nowlen, 6413
MS0405	T. D. Brown, 12333	MS0748	C. P. Ottinger, 6413
MS0405	S. E. Dingman, 12333	MS0748	D. G. Robinson, 6413
MS0434	R. J. Breeding, 12334	MS0748	H. O. Whitehurst, 6413
MS0451	J. E. Nelson, 6515	MS0748	T. A. Wheeler, 6413
MS0451	L. A. Snyder, 6515	MS0748	D. W. Whitehead, 6413
MS0451	D. C. Barton, 6515	MS0759	I. G. Waddoups, 5845
MS0490	J. A. Cooper, 6252	MS0759	M. K. Snell, 5845
MS0615	R. L. Perry, 6252	MS0779	J. C. Helton, 6849
MS0615	R. D. Hartman, 6252	MS0828	M. Pilch, 9133
MS0717	R. L. Craft, 6514	MS9018	Central Technical Files, 8945-1
MS0736	T. E. Blejwas, 6400	MS0899	Technical Library, 9616 [2]
MS0744	F. J. Wyant, 6420	MS0612	Review and Approval Desk, 9612, for DOE/OSTI [1]
MS0746	D. J. Anderson, 15312		
MS0746	L. Swiler, 15312		
MS0746	R. D. Browitt, 15312 [2]		
MS0747	A. L. Camp, 6410		
MS0747	R. S. Campbell, 6410		
MS0747	R. G. Cox, 6410		
MS0747	V. J. Dandini, 6410		
MS0747	S. L. Daniel, 6410		
MS0747	F. A. Durán, 6410 [10]		
MS0747	J. LaChance, 6410		
MS0747	T. T. Sype, 6410		
MS0747	G. D. Wyss, 6410 [10]		
MS0748	R. D. Waters, 6413		
MS0748	C. B. Atcitty, 6413		